Universidade do Minho
Mestrado Integrado Engenharia Eletrónica Industrial e Computadores
(MIEEIC)

Embedded Systems

---

# REMOTE CONTROLLED SONAR CAMERA

---

-Project Analyse-

Authors:
João Sousa nº82273
Pedro Sousa nº82041

Professor:
Adriano Tavares
José Mendes

November, 2019-2020

# Contents

# List of Figures

# List of Tables

## I. Problem Statement

The impact of technology evolution on humanity has increased very fast. The use of technology is spreading through all the tasks of our professional and personal life. As so, assuring our security and safety it’s one of our main concerns.

With our project we intend to improve the cameras of today. For example, most car’s back cameras, used for parking, can’t measure distance precisely and that could result into serious problems. We design to make a remote-controlled camera that can measure distances accurately of close objects. That's why we call our project "RCSC"- Remote Controlled Sonar Camera.

Our project would improve not only car’s cameras, but in robots it could be used for a good environment view and object collision.

## 1. Problem Analysis

The problem above described needs to be deconstructed and analyzed in more detail. It is mandatory to define Entities, as well as their functions and correlations.

After the statement core of our problem, we encounter 5 main subsystems: the "Device System", the "Remote Station", the "Sensors & Actuator" the "User" and the "Interface".

- The "User" views the video transmitted and remote controls the motors through the "Interface",
- The "Interface" mainly shows the video, the radar and the controls to the motors. It receives and sends data to the "Remote Station".
- The "Remote Station", works as a bridge of ethernet communication and data storage between the "Device System" and the "Interface", increasing the project flexibility.
- The "Device System" is the main processor and controller. Receives data from "Sensors" and sends it to the "Remote Station". Receives from the "Remote Station" the defined controls set by the "User" and based on them actuates on the "Motors"

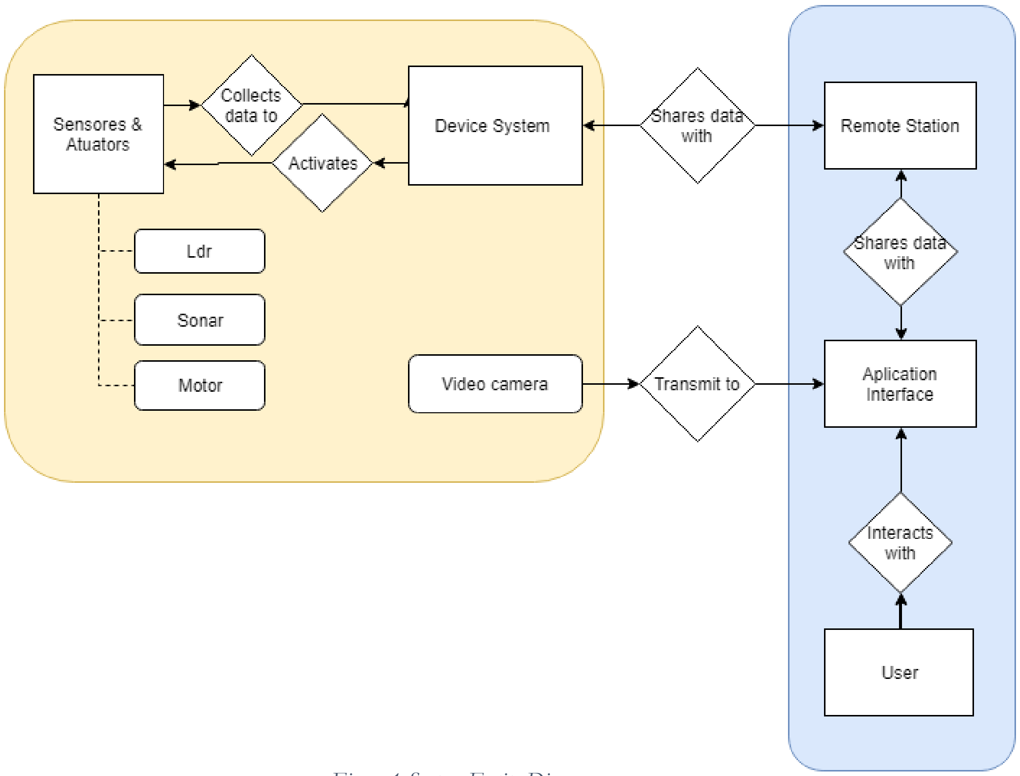


*Figure 1-System Entity Diagram*

# II. Market Research

The increasing demand for surveillance systems to enhance safety and security are increasing, and features such as a WIFI camera with sonar gives our application a huge potential.

However, there is a vast presence of similar and very competitive products in the market that represents a big threat to our product.
In the end, our product could not be as efficient in some specs as those advanced security cameras but could potentially improve the streaming methods of today.

## 1. Target Market

Our application aims for the need for a simple control of a personal camera, that could be easily be implemented by the consumer.
It would aim for the public who still wants a remote-controlled WIFI IP Camera, but that don't want it to be fixated to the company and camera limitations. In our product, the consumer could fit any camera in the control station and directly connect the camera to the system of the interface.

It could be a major improvement in the lives of streamers, entertainers, producers and make jobs in filming way more uncomplicated.

## 2. Competitive Analysis

- **INQMEGA PTZ Camera**
  This camera provides an IP/wireless network connection and has a horizontal and vertical tilt that can be remoted via phone/Ipad/PC. It's one of the most affordable budget outdoor IP security cameras on the market, with the price of 77$.

*Figure 2 -INQMEGA Camera*

There are many similar products to the one above in the market, varying in price, performance or quality. Not only products, but many DIY (Do It Yourself) projects can we found available to the public, such as:

- **Instructables - Security Camera (1)***

  **-**DIY Security Camera project, which takes a picture of the person at the door and sends an email with the picture. Using only Arduino, an PIR sensor and a USB Webcam.
  However, compared to our product, this project is way simpler and with less features.

- **LinusTechTips – WIFI* Security Camera (2)***

  -DIY WIFI Security Camera, which provides a full guide for video streaming to a hub server using Raspberry Pi. Our product puts himself in the same level of complexity as this project, but with more features, making our product more authentic.

# III. Analysis Phase

## 1. System

### 1.1 System Overview

Based on the analysis above the project will be composed by 1 microcontroller, 3 sensors and 1 actuator:

- The STM32 – The main controller of the application. It gathers data from the sensors, activates the motors and communicates with the remote system. Reads from the Remote Station Server the data inputted by the User and based on them controls the motors.
- The Sensors – collecting useful data for the System
    - Sonar - to radar scan the environment;
    - LDR - detect day or night through light;
    - Camera - filming video (showing on the interface) of the environment;
- The Motors – 2 servomotors that allows to move the camera/sonar (coupled to the motors) in a horizontal and vertical axis.

The Microcontroller after gathering the data from sensors, sends it to the remote station through the help of ethernet dedicated libraries. It also receives from the remote station the data to control the motors.

As so, there is an external application running the Interface that allows the User to remote control the camera/sonar direction and shows a radar of the environment.

However, due the system limitation, the camera must be directly connected to the device of the interface for the video sharing.

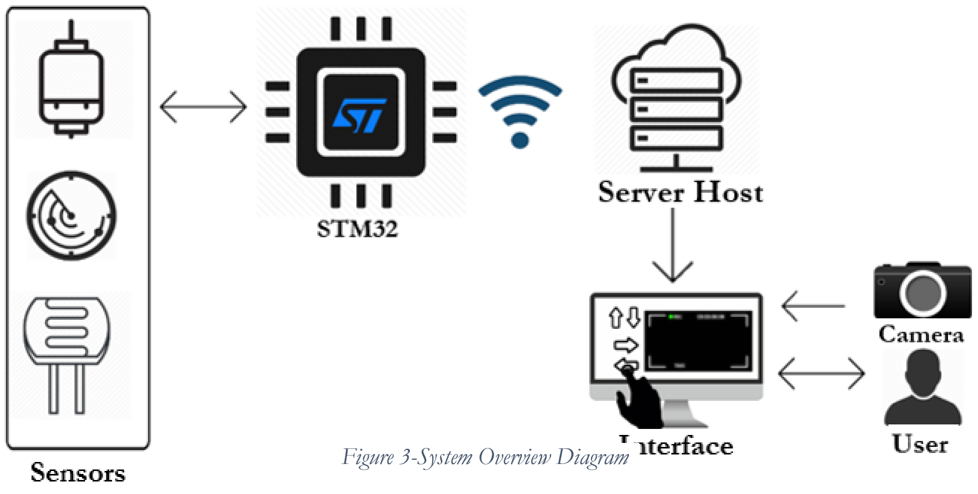


*Figure 3-System Overview Diagram*

# 2. Requirements and Constraints

In order to the better predict a good performance from the final product, its mandatory to establish all the requirements and constraints implied.

## 2.1 Non-Technical Constraints

Represents all the limitations that exist apart from the practical development of the project. In this case, the project group must:

- Not surpass the final deadline;
- Maximum of two students per group;
- Be budget friendly duo to money limitations;

## 2.2 Technical Constraints

Represents all the established limiting factors that affects the execution of a project. In this project, its implementation requires:

- STM32F767ZI;
- Real time system – FreeRTOS;
- Keil MKD-ARM (uVision) as Run-Time Environment;
- 3 non-identical sensors;
- Machine-Learning and/or adaptive control;

## 2.3 Functional Requirements

It defines all the specific functionality that defines what the system will accomplish. As so, this project must:

- Scan the environment;
- Move the camera and sonar direction;
- Communicate with remote station server;
- Allow the User control of the direction;
- Display radar scan and video;

## 2.4 Non-Functional Requirements

It defines all the quality attributes we pretend our system to have. As so, we design it to be:

- Friendly user and coherent interface;
- High flexibility;
- Low cost;
- Sturdiness;

# 3. System Architecture

As any other good project, it starts on a good fundamental well-constructed Architecture. Making the connection between the hardware and the software, is the essential key to achieve a certain state of quality.

## 3.1 Hardware Architecture

The System hardware consists on a station of sensors connected to the STM32, the main processor/controller. The STM32 connected indirectly to the interface, where they exchange data. The interface allows the control of the motor (changing the camera/sonar direction) and displays the sonar inputs and the video.

The diagram in figure X shows a simple representation of the Hardware.

## 3.2 Hardware Specification

- STM32F767ZIT6 board – The main controller of the application for data processing and data transmission:
  - Arm Cortex-M7 32-bit RISC with FPU.
  - Chrom-ART AcceleratorTM.
  - Up to 2 Mbyte of Flash memory and 512 Kbytes SRAM.
  - Up to 168 I/O ports.
  - Up to 18x timer, 4x I2C interfaces, 3x ADCs and 2x DACs: 12 bits each.
  - Up to 6 SPIs, 4 USARTs/4 UARTs and 2 Serial Audio Interfaces (SAI).
- Light sensor – to get the light luminosity of the environment. Simple GPIO implementation;
- Servo-Motors – to aim the camera to the desired location. The system can read the motors to know its angular position;
- Camera – for capturing image/video through USB connection.

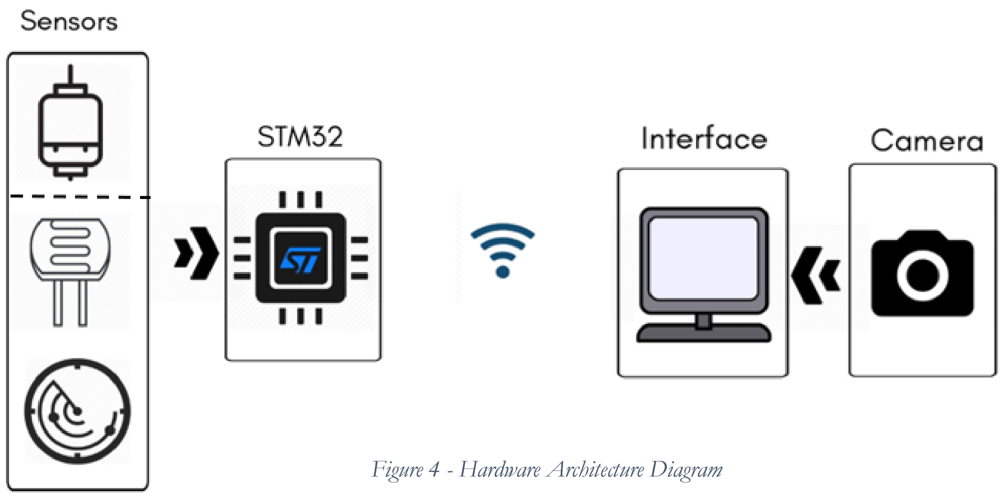


*Figure 4 - Hardware Architecture Diagram*

## 3.3 Software Architecture

In terms of the software, our system requires some specific drivers for the STM32 establish communication with the sensors and the remote station, and to actuate on the motors. The Middleware covers the acquisition and processing of the data and the communication through specific ethernet libraries. In Application there is the Graphic Interface showing all the data gathered, and also the control of the Camera/Sonar Tower.

The diagram in figure 4 shows a simple representation of the Software.

## 3.4 Software Specification

- FreeRTOS – Real time Operation System;
- Keil µVision - programming IDE;
- C and C++ programming language
- Qt creator - for build the interface;

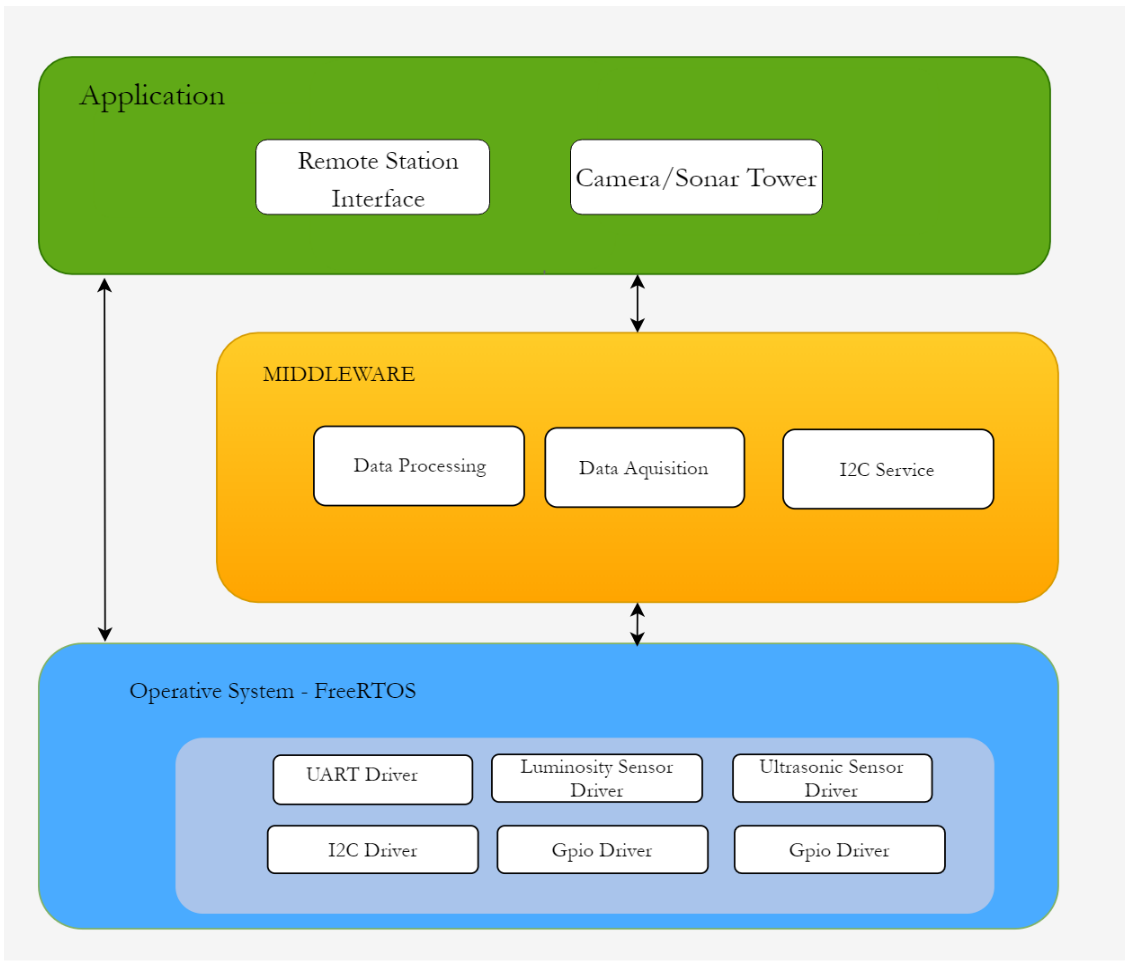


*Figure 5 - Software Architecture Diagram*

# 4. Local System

## 4.1 Systems Events

The board below presents all the events that can be triggered in our system, it is divided by name, description, source and type. The table above shows how many synchronous and asynchronous events we have, that means respectively they are executed at the same time or not.

The events that need to be synchronous are: "Sensor Data Gathering", "Access Request", "Data Transfer", "Data Analyses" and "Streaming Camera". These events represent the prime features of our project.

In opposite, rest of the events are asynchronous because their execution isn't predictable as they only happen when the User decides to interact with the Interface. The events are: "Push button" and "Choose Operation Mode". They establish the mode of operation and the control of the camera movement.

| ***Event Name*** | **System Response** | **Source** | **Type** |
|---|---|---|---|
| *Sensor Data Gathering* | Collects data from sensors. | Time | Synchronous |
| *Access Request* | Establish the connection between remote system and local system | Local System | Synchronous |
| *Data Transfer* | Establish communication between host-client. Transfer data between webpage and local system. | Time | Synchronous |
| *Data Analysis* | Process the data collected and make important decisions | Local System | Synchronous |
| *Push button* | Sends command to local System | User | Asynchronous |
| *Choose Operation Mode* | Defines if Automatic or Manual mode | User | Asynchronous |
| *Streaming Camera* | Put the video of the camera on interface | Camera | Synchronous |

*Table 1 - Table of Events*

## 4.2 Use Cases Diagram

In our system, RCSC (Remote Controlled Sonar Camera), there are two primary actors (the User and Time) and three secondary actors (Motors, Camera and Sensors). They are going to be responsible to achieve the goal of our system. User must be capable to put the program on automatic or manual mode
The System also has the possibility of an adaptive control, meaning two different modes:

- **Manual mode:**

As we see bellow on diagram, when the user interacts with the interface, the system receives commands from webpage application to control the motors allowing the user for the manual analysis.

With the triggering of Time, the system samples camera image and reads the environment. After that, the system groups the collected information, processes it and sends it to the webpage application.

The only effect of the User in the system is to move the camera angle through control of the Motors.

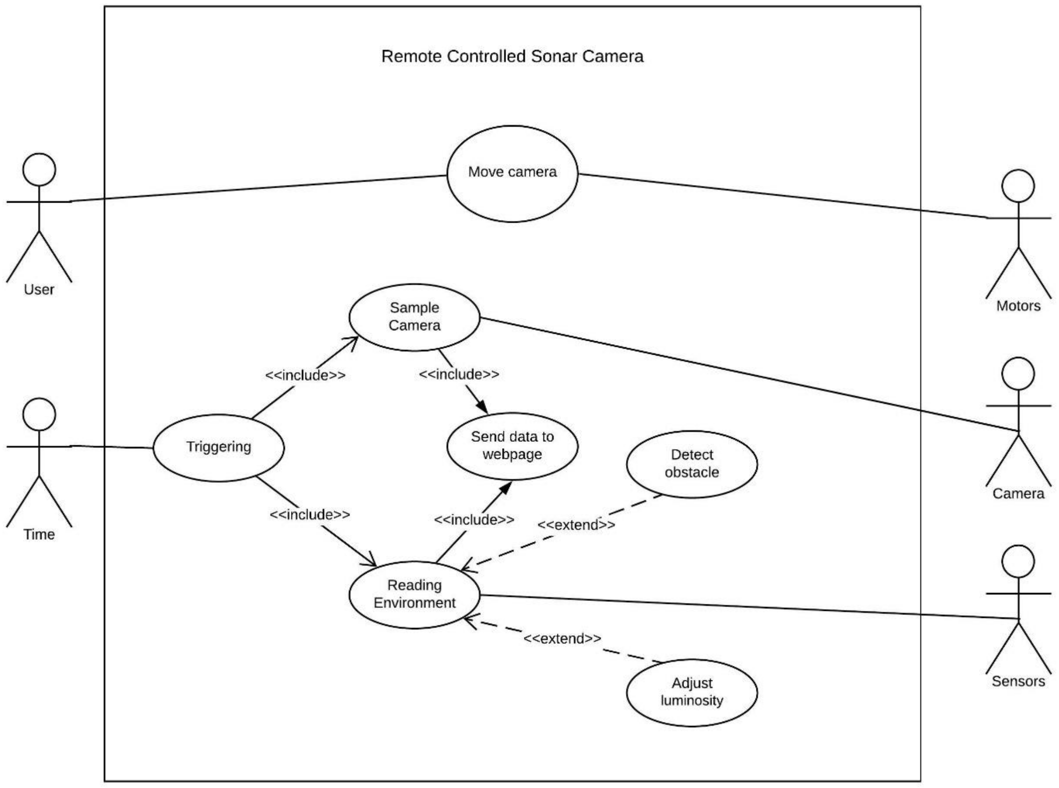


*Figure 6 -System Use Case Diagram in Manual Mode*

- **Automatic mode:**

The automatic mode differs in the control of the movement of the camera. Here, the system is autonomous, where it scans the environment and adapts the position of the camera pointing to the closest object.

In this diagram, the User no longer makes part as one of the actors, relying only on the rest of them the good performance of our system.

Here, Time is the principal actor, causing the trigger to read the sensors and read the sample video from camera. When the Sonar Sensor detects an obstacle, it performs an adaptive control with the information gathered, operation on the motors to make the camera follow the obstacle. Then, when all information is processed, the local system sends it to the webpage application.

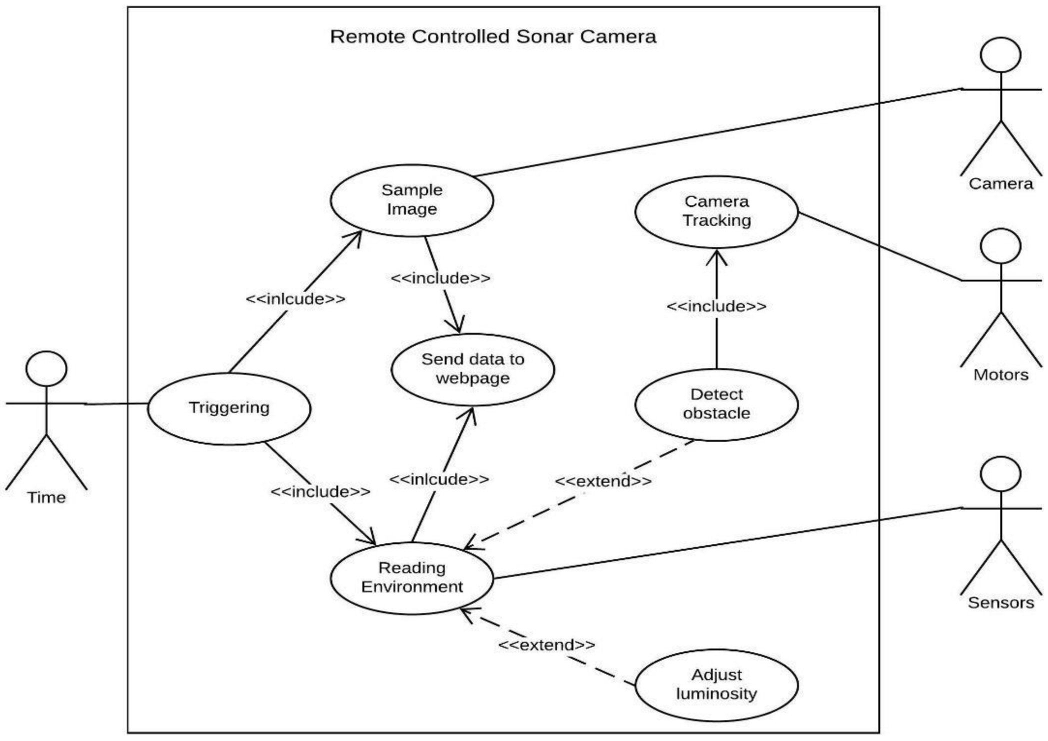


*Figure 7 - System Use Case Diagram in Automatic Mode*

## 4.3 State Chart Diagram

Our main system can be represented by the diagram below too. With this diagram it's possible to better understand the flow of our program, state by state.

First, it initializes the configuration of the system (one of the most important states). Next, the system collects on a constant period the data from the sensors to get the real-time environment vision, when this state is complete data will be analyzed and processed. At the "Process" state, the system will send the processed information and activate the motors.

When the system is on manual mode, where the User interacts with the interface, it will analyze the inputted command and move the motor based on the remote control made by the User.

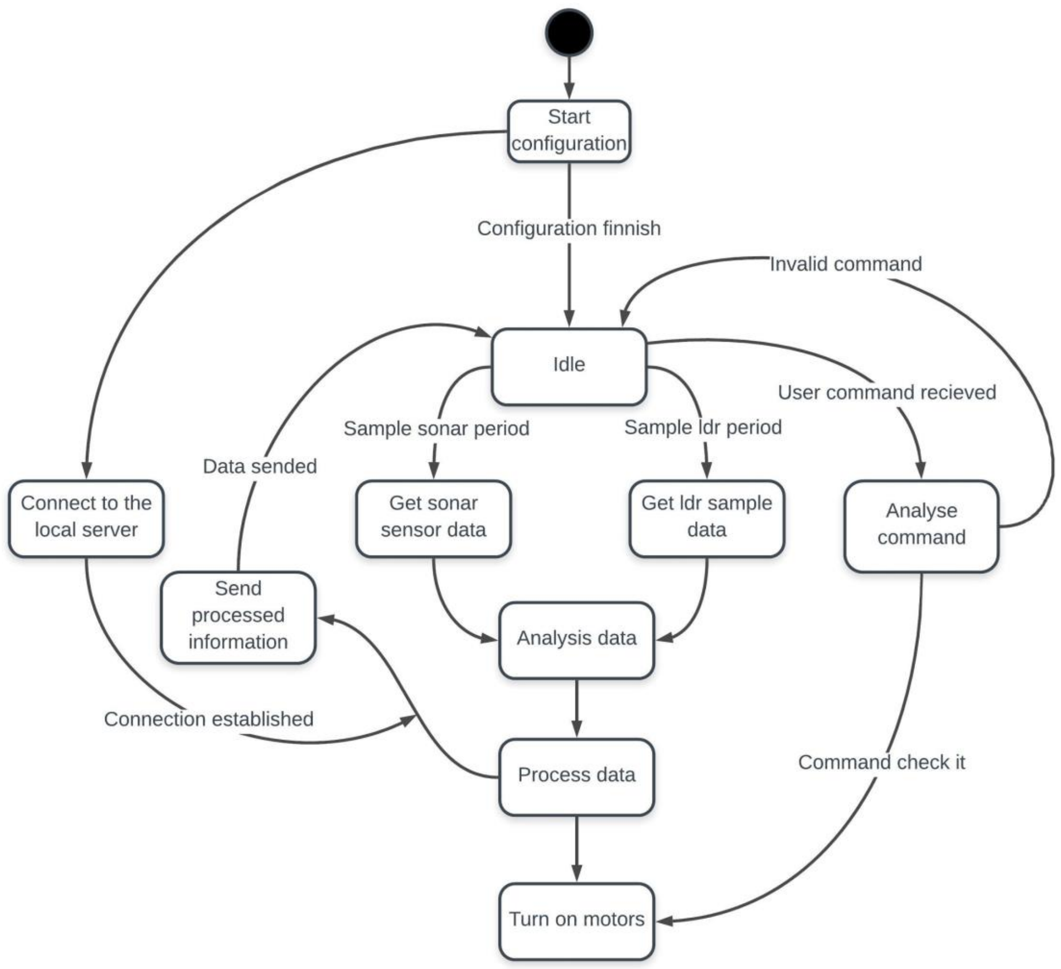


*Figure 8 -System State Chart Diagram*

## 4.4 Sequence Diagrams

As shown in use cases diagram above, “Time” is the main actor on the device system. On sample period, that it's triggered by “Time” (synchronous event), it requests sample of the sensors (sonar, ldr and motor position) and streams the camera video.

The Local system request samples to the sensors and if the response is accepted the data will be analyzed, but if was not, the “Local system” will request again until the answers be acceptable.

After collecting and analyzing the data, the system request access to communicate with “Remote System” (webpage) and sends the information to it and. If the system is in Automatic mode, it will adjust the angle of the camera based on the analyzed data

.

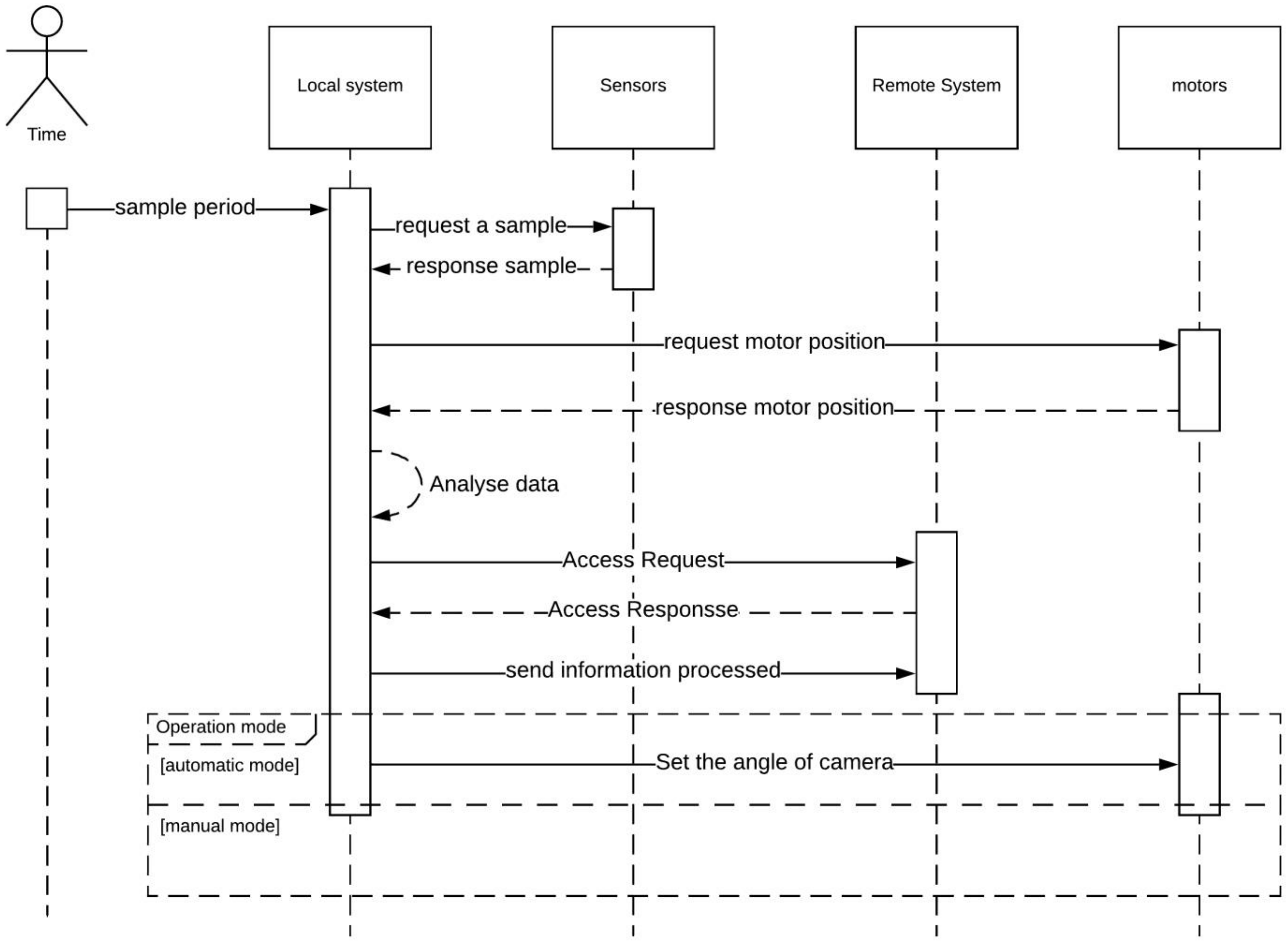


*Figure 9 - System Sequence Diagram with Time as Actor*

With this second diagram we pretend to show what happened when the User interacts with our system (asynchronous event). When the user press on one of the direction buttons (UP, DOWN, LEFT or RIGHT) of the interface, the "Remote system" request access to the "Local system" and sends the "user command". Then the local system translates the command to activate the servo-motors to move the camera in de choosen direction.

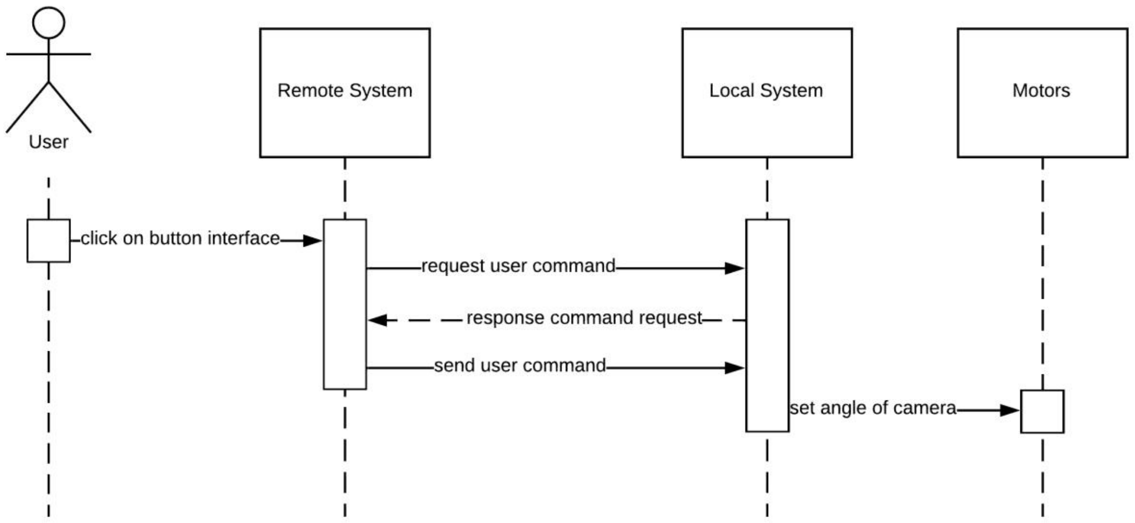


*Figure 10- System Sequence Diagram with User as Actor*

Other process is the streaming of the video. There is refresh rate period in which the remote system request to the camera the video frame, to show on the interface.

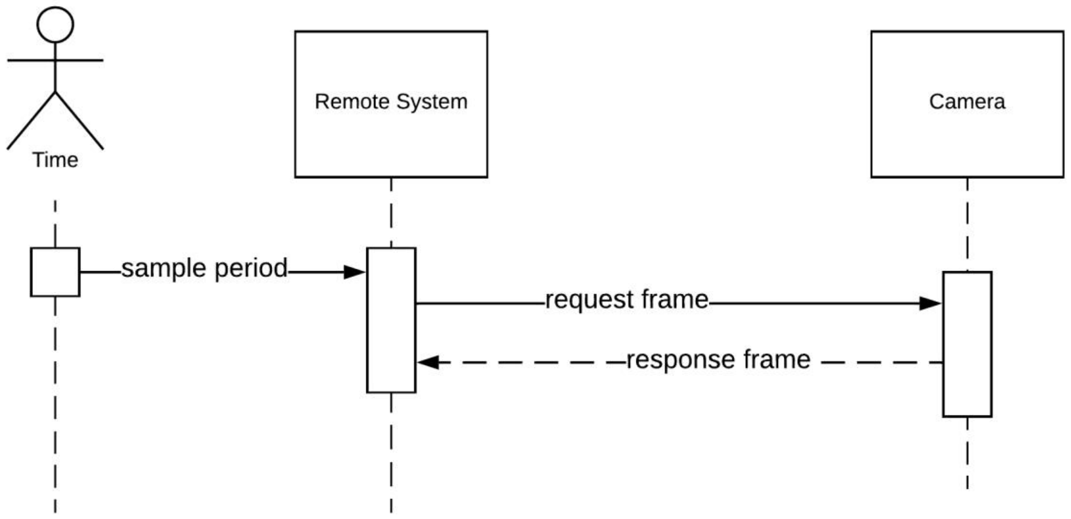


*Figure 11 - Video streaming Sequence Diagram with Time as Actor*

# IV. Design Phase

On this phase our project we decrypt the fundamentals that was described on last phase, Analysis phase. We intend project the product in detail.

At first, we going to introduce a several base on theoretical concepts that are important to understand some contents covered on this project.

With the learned subject of class, we intend to meticulously cover all of software and hardware topics and respect its features.

## 1.Hardware Specification

### 1.1 STM32F767ZIT6 board

Our main development board is the STM32F767ZI and has an ARM®32-bit Cortex®-M7 CPU with an FPU (floating point unit), able to speed up the calculations, improving the processing time required to the operations. The STM32F407VGT6 has the following key features:

• Core: ARM®32-bit Cortex®-M7 CPU with FPU, 216 MHz max CPU frequency;
• 2 Mbyte of Flash memory;
•512 Kbytes SRAM;
• On-board ST-LINK/V2;
• Low-power operation;
• General-purpose DMA: 16-stream DMA controller with FIFOs and burst support;
• 17 timers;
• 140 I/O ports;
• 3 ADCs (with 12-bit resolution);
• 4 I2C;
• DSP.

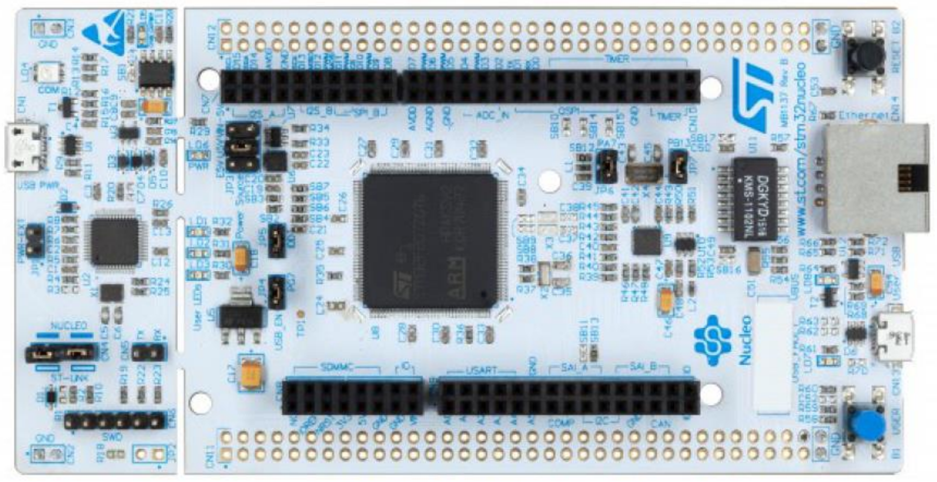


*Figure 12 - STM32F767*

## 1.2 OV7670 Camera Module

The OV7670 image sensor provides all functions of a single chip of VGA camera and image processor. Through SCCB bus control, the sensor can output the whole frame, sampling, and various resolution 8 bits of data. The product VGA image can reach up to a maximum of 30 frames per second. Users can completely control the image quality, data format and transmission mode. All the process of image processing functions can be through the SCCB programming interface, including gamma curve, white balance, saturation and chroma.

Parameters and Features:

- Photosensitive array: 640X480
- IO Voltage: 2.5V to 3.0V (internal LDO for nuclear power 1.8V)
- Power operation: 60mW/15fpsVGAYUV
- Sleep: <20μA
- SNR: 46 dB
- Dynamic range: 52 dB
- View Mode: Progressive
- High sensitivity suitable for illumination applications
- Low voltage suitable for embedded applications
- Standard SCCB interface compatible with I2C interface
- RawRGB, RGB, YUV and YCbCr output format
- ISP has a compensation function to eliminate noise and dead pixels

The following schematic diagram show a basic camera-based system. The camera module is powered from a single +3.3V power supply. An external oscillator provides the clock source for camera module XCLK pin. With proper configuration to the camera internal registers via I2C bus, then the camera supply pixel clock (PCLK) and camera back to the host with synchronize signal like HREF and VSYNC.

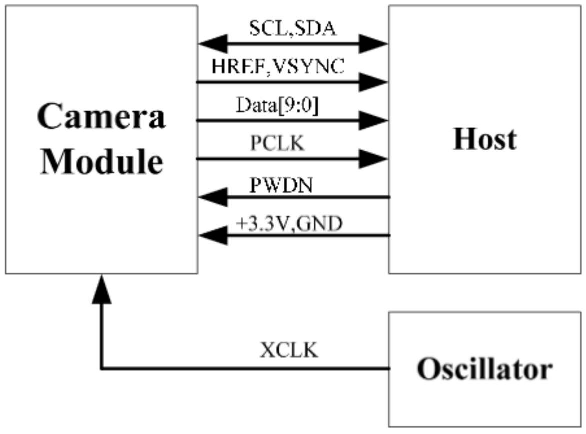


*Figure 13- camera module connection diagram*

**Interface**

| OV7670 | STM32F7 Function Pin | Board Pin |
|---|---|---|
| **3V3** | 3V3 | 3V3 |
| **GND** | GND | GND |
| **SCL** | I2C_scl | PB8 |
| **Sdata** | I2C_sdt | PB7 |
| **VSYNC** | GPIO Pin | PD5 |
| **HREF** | GPIO Pin | PD4 |
| **PCLK** | SPI3SCLK | PC10 |
| **XCLK** | --- | --- |
| **D1-D7** | GPIO Pin | (7 Digital input pins) |

*Table 2- Cemera Module interface*

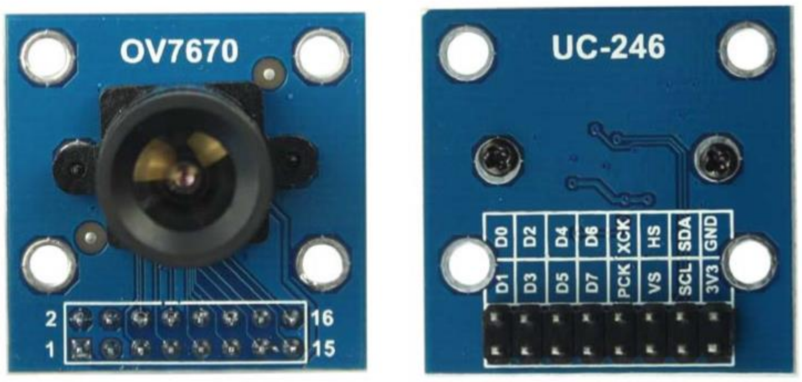


*Table 3- OV7670*

**Test cases**:
To have a good use of the camera it is necessary to be able to perform some tests. Such as, take a picture to see the resolution and other features of the image, capturing an image checking if was stored and test a video for streaming.

| Test Cases | Expected Output | Real Output |
|---|---|---|
| **Take a picture** | An image is shown on screen; | |
| **Capture an image** | Stored pictured that was captured; | |
| **Test Video** | Streaming a video; | |

*Table 4- Camera module test cases*

## 1.3 Servo Motors

The 2 servo motors allow the angle adjustment of the camera in order to follow a person, rotating in the two axis (horizontal and vertical axis). The SONICMODELL Servo Motor is a tiny and lightweight motor with high output power and has metal gears for added strength and durability.
The servo has three wires. Two are for power and the third wire is for signals to position the servo. The signal wire expects input from a pulse width modulator.
The period should be 20 milliseconds long and the duty cycle encodes the position of the motor. If the duty cycle is 1.5 millisecond, the servo is positioned at 0 degrees. If the duty cycle is 2 milliseconds, is all the way to the right (90 degrees). If the duty cycle is 1 millisecond, the servo is all the way to the left (-90 degrees).

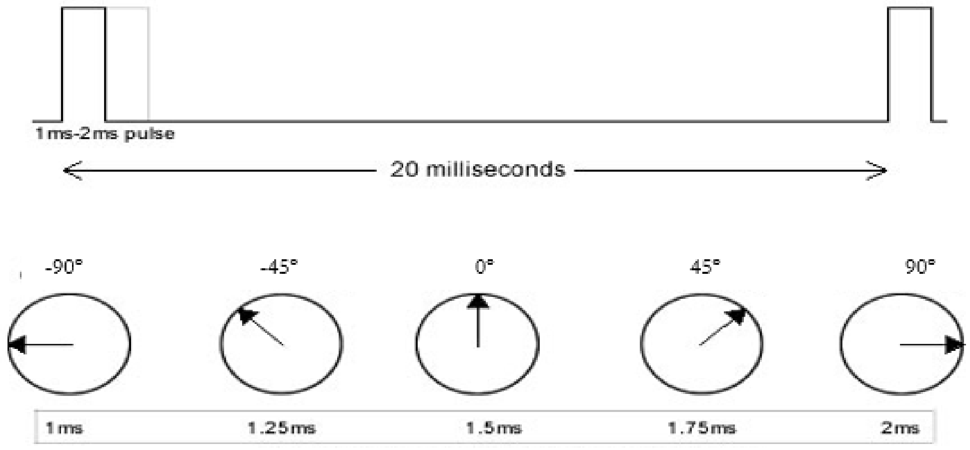


*Figure 14- PWM period cycle*

**Specifications:**

- Weight: 18 g
- Dimension: 6x9x3 cm
- Stall torque: 1.8 kgF·cm (4.8V), 2.2 kgf·cm (6 V)
- Operating speed: 0.1 s/60 degree(4.8V), 0.08s/60 degree(6 V)
- Operating voltage: 4.8 V – 6.0V
- Dead band width: 5 µs

*Figure 15- Servo Motors*

## Test Cases:

To have a good performance of the motors it is necessary to be able to perform some tests. Such as, initialization **PWM** to see the output waves, **driving motors to several directions** checking which duty-cycle percentage correspond to each angle.

| Test Cases | Expected Output | Real Output |
|---|---|---|
| **Initialization PWM** | visualizing the wave; | |
| **drive motors to several directions** | input different duty-cycles of PWM | |

*Table 5 - Motors test case*

**Interface**

| Servos | STM32F7 Function Pin | Board Pin |
|---|---|---|
| **VCC** | 3V3 | 3V3 |
| **GND** | GND | GND |
| **Signal** | Timer3_pwm | PA6 |

*Table 6- Motors Interface*

In order to provide a signal with enough current, and also in terms of protection, it will be implemented a circuit, as in the diagram bellow, with a N-Type Mosfet (working as the interrupt, the resistor and capacitor work as regulators).

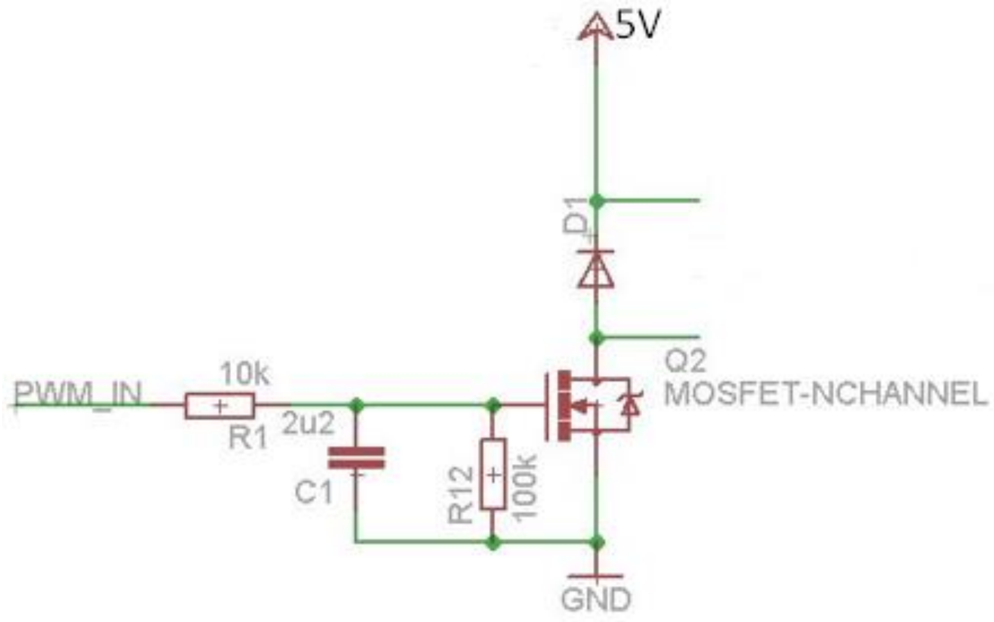


*Figure 16 - Mosfet Protection schemantic*

## 1.4 DC POWER and Battery

For powering the STM32F7 a 5V DC Powerbank is used. The camera and the sensors are then powered by the board. The motors powered by another 5V DC converter to avoid current overload and damage the board.
In order to provide enough current to the board and the sensors the board must at least have 2A:

- Output voltage: 5Vdc voltage and capacity up to 2500mAh;2 USB outputs;
- 2 USB outputs (only one is necessary)

Figure 17- Powerbank

This is a simple AC-DC converter that has an alternate current (AC)input tension of 230 V from the power grid and a direct current (DC) output tension of 5V. The AC-DC that we will use has the following characteristics:

• Input voltage: 100-240Vac and between 50/60Hz;
• Output voltage: 5Vdc voltage and capacity up to 2500mAh;
• Protection against overload and over voltage.

Figure 18- AC/DC converter

## Test cases

| Test | Expected output | Real output |
|---|---|---|
| Powerbank | Powerbank provides the mentioned power | |
| AC/DC converter | DC Power provides the mentioned power | |

Table 7- Power sources test case

## 1.6 Luminosity Sensor – LDR

The system will collect data about the ambient luminosity, and to do so, it will use this Light sensor module based on photodetector GL5528 to detect the light intensity of the environment. As the resistance of the sensor varies depending on the amount of light it is exposed to, the output voltage changes with the light intensity. This module outputs both analog and digital signals, which can be used to trigger other modules. Also, the potentiometer can be used to adjust the sensitivity of digital output. The digital terminal output HIGH when the light intensity exceeds the value set by potentiometer and vice versa.
The outputs of analog terminal increase with the light intensity.
Features:

- Working voltage: 5V
- Output signal: Analog and Digital;

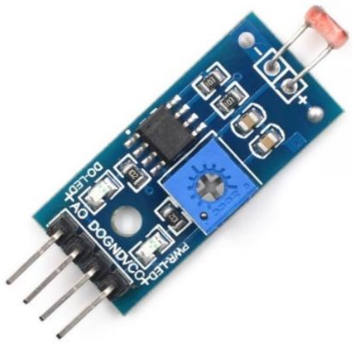


*Figure 19- LDR module*

**Test Cases**

| LDR | Expected Output | Real Outp |
|---|---|---|
| No light | Vout between 3v and 2.57V and R between 35k and 100k (alterar valores) | |
| Medium light | Vout between 2.2V and 1.36V and R between 10k and 20k | |
| High intensity light | Vout between 0.8 and 0.5V and R between 1k and 3k | |

*Table 8 - LDR test cases*

### Interface:

| LDR | STM32F7 Function Pin | Board Pin |
|---|---|---|
| **VCC** | 3V3 | 3V3 |
| **GND** | GND | GND |
| **A0 (analog pin)** | GPIO PIN | PD6 |
| **D0 (digital pin)** | - | - |

*Table 9 - LDR interface*

## 1.7 HC-SR04 Ultrasonic Sensor

The sonar uses sound waves to measure the distance of a obstacle. To implement object detection and to measure distance we will need a HC-SR04 Ultrasonic Sensor module.

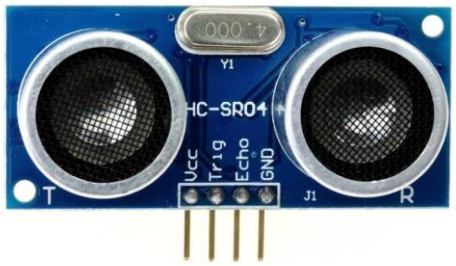


*Figure 20 - HC-SR04 sensor*

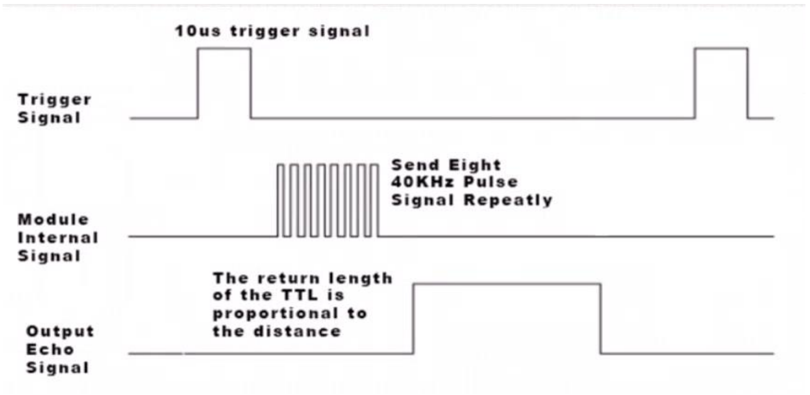
**Specifications:**

- Power supply: 5V DC
- Operation current: 2mA
- Effect angle: 15º
- Range: 2cm – 4m
- Precision: 3mm

It is composed by a transmitter and a receiver. The transmitter sends ultrasonic sensor with the specific sensor frequency, non-audible for humans, and part of this wave will be reflected in an object and retrieves it back to the receiver.

With this mechanism we can measure the time that the wave spends until achieve the obstacle and back. After that, we can finally, by software calculate the distance, we just need to multiply the time by the velocity of sound in air (340m/s) and split all by two, since wave go through the same distance two times.

To enable the measurement, it´s necessary to put in High state the pin trigger for ten seconds. After the module emits a signal, when an object is detected and the signal reflected, the pin ECHO will be in High state for a period proportional to the distance between the sensor and the obstacle.


10us trigger signal
Trigger Signal
Send Eight 40KHz Pulse Signal Repeatly
Module Internal Signal
The return length of the TTL is proportional to the distance
Output Echo Signal
*Figure 21 - Sonar pins signal*

### Test case

| Test | Expected output | Real output |
|---|---|---|
| Connect to the Board | The Board successfully reading the sonar values | |
| Detect Object | Detect an interference in the signal | |
| Measure distance | Retrieve the exact value of the object distance | |

*Table 10 - SSonar test case*

### Interface:

| Ultrasonic Sensor | STM32F7 Function Pin | Board Pin |
|---|---|---|
| **VCC** | 3V3 | 3V3 |
| **GND** | GND | GND |
| **Echo** | GPIO PIN (OUTPUT) | PD3 |
| **Trigger** | GPIO PIN (INPUT) | PD1 |

*Table 11 - Sonar interface*

## 1.8. Esp8266 module

In order to stabilish a remote communication with our system we need a module that be responsible for that. The ESP8266 module, it is a self-contained SOC with integrated TCP/IP protocol stack that can give any microcontroller access to our WiFi network.

The ESP8266 is capable of either hosting an application or offloading all Wi-Fi networking functions from another application processor. Each ESP8266 module comes pre-programmed with an AT command set firmware, meaning, you can simply hook this up to your Arduino device and get about as much WiFi-ability as a WiFi Shield offers.

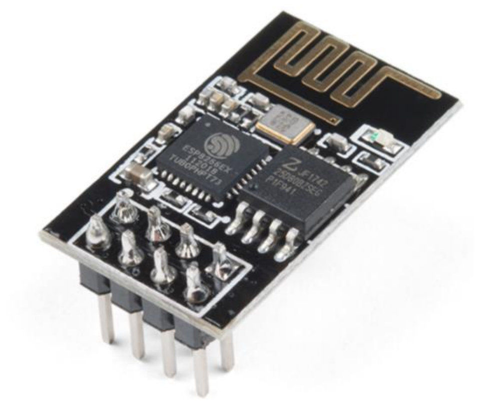


Figure 22 - WiFi Module

**Interface:**

Here we described how is composed the connections between this module and the STM32:

-GROUND – Used for connected to the ground of the circuit.
-TX – Used to connected to Rx pin of STM32 to upload program.
-RX – Receive data bit.
-GPIO-2 – Used for a general-purpose Input/Output pin.
-CH_EN – Used to chip enable – Active high.
-GPIO-0 –It takes module into serial programming when held low during start up.
-RESET – Used for resetting the module.

| ESP8266 | STM32F7 Function Pin | Board Pin |
|---|---|---|
| **VCC** | 3V3 | 3V3 |
| **GND** | GND | GND |
| **TX** | Rx | PB12 |
| **RX** | Tx | PB13 |
| **GPIO-2** | GPIO PIN (INPUT) | PD7 |
| **GPIO-0** | GPIO PIN (INPUT) | PD8 |
| **RESET** | --- | ---- |
| **CH_EN** | ---- | ---- |

Table 12 - WiFi Interface

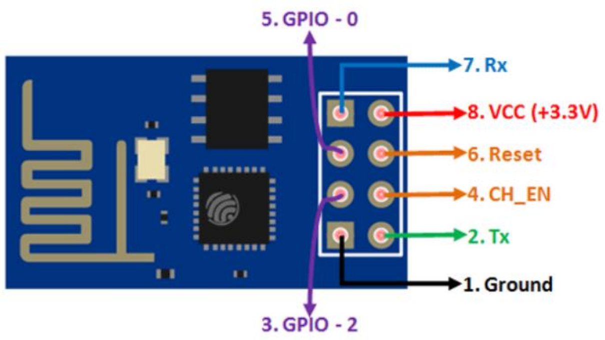


*Figure 23- WiFo module pinout*

**Test Cases:**

| Esp8266 | Expected Output | Real Outp |
|---|---|---|
| Test the connection | Stablish a connection between two devices. | |
| Test data transmission | Send a byte. | |
| Test data reception | Receive a byte. | |

*Table 13 - WiFi teste cases*

## 1.9 Pan & Tilt

The structure of the application we based on a Pan-Tilt model, where the two servo-motors will be coupled. It will enable the orientation of camera in 2 axis, necessary for the face centering of a person (aiming the camera in the person direction). Specifications:

- PWM driver: PCA9685
- PWM resolution: 12-bit
- Ambient light sensor: TSL2581FN
- Ambient light resolution: 16-bit
- Communication interface: I2C
- Operating voltage: 3.3V/5V
- Dimension: 56.6mm x 65mm

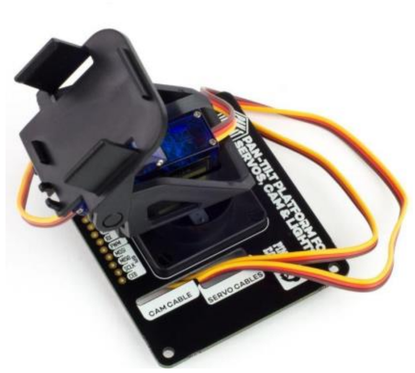


*Figure 24- Pan & Titlt*

**Test Cases**

| Test Cases | Expected Output | Real Output |
|---|---|---|
| **Test mechanism** | visualize the movements | |
| **Test Tilt** | input PWM in tilt servo-motor | |
| **Test Pan** | input PWM in pan servo-motor | |

*Table 14 - Pan & Tilt test case*

**Interface**

The communication can be performed by I2C. However, it's possible to use the individual pins to control the motors individually, and the light sensor will be implemented with a different module. So, the I2C interface pins will not be connected to the board.

| Pan & Tilt | STM32F7 Function Pin | Board Pin |
|---|---|---|
| **VCC** | 3V3 | 3V3 |
| **GND** | GND | GND |
| **S1** | Timer2_pwm | PA0 |
| **S2** | Timer3_pwm | PA6 |
| **SDA** | --- | --- |
| **SCL** | --- | --- |
| **INT** | --- | --- |

*Table 15 - Pan & Tilt Interface*

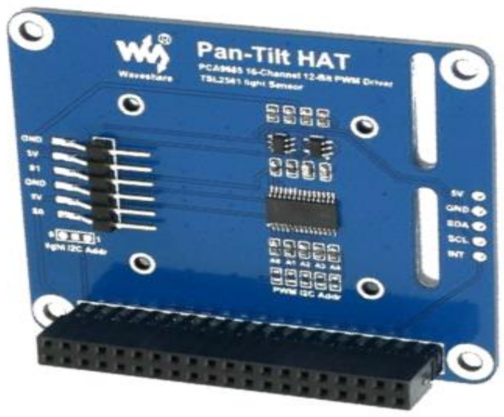


*Figure 25 - Pan & Tilt Pinout*

## 1.10 Structure

In order to compact all the components in one structure with all the required software, it was designed the following 3D model. In the bellow area is is compacted the STM board, the sensors and the Power supply. It shall have all the entries necessary for the cabling, cooling and have enough visibility for the sensors. In the above part there is the tilt and pan structure for the direction control of the camera.

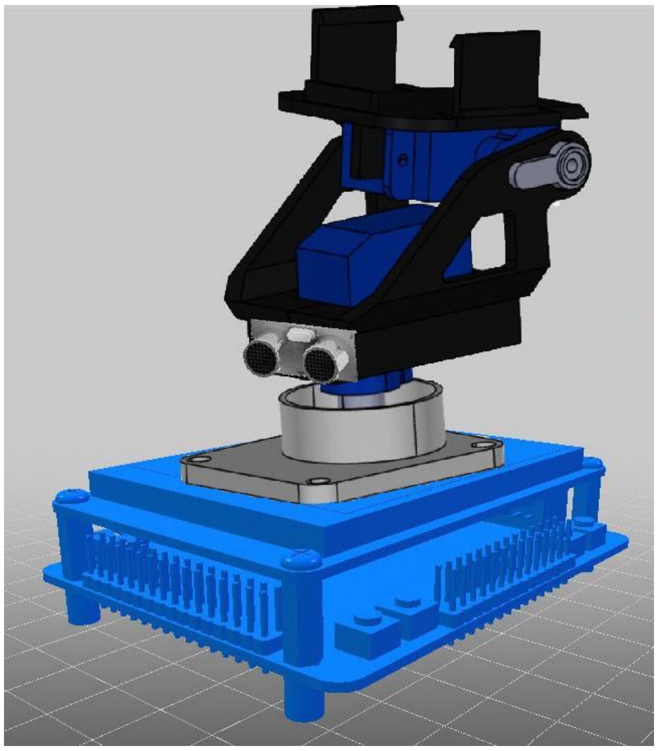


*Figure 26 - Model Structure*

## 2. Hardware Peripherals and Communication Protocols

These protocols are responsible for the interaction and the stability of the communication between the hardware and our board (STM32). For different hardware, in our case, we have different protocols, for instance, the local system has 1 UART, 1 DCMI, 2 PWM, 1 GPIO and 1 ADC. The UART is used to receive and send information via wifi with the WiFi module. The DCMI protocol is used to connect a parallel camera module to the STM32. In order to retrieve information from two analog devices a ADC is needed as it is possible to see in the LDR. The Sonar works with a digital function that is connected to a GPIO to warn the local system the distance of the detected object.

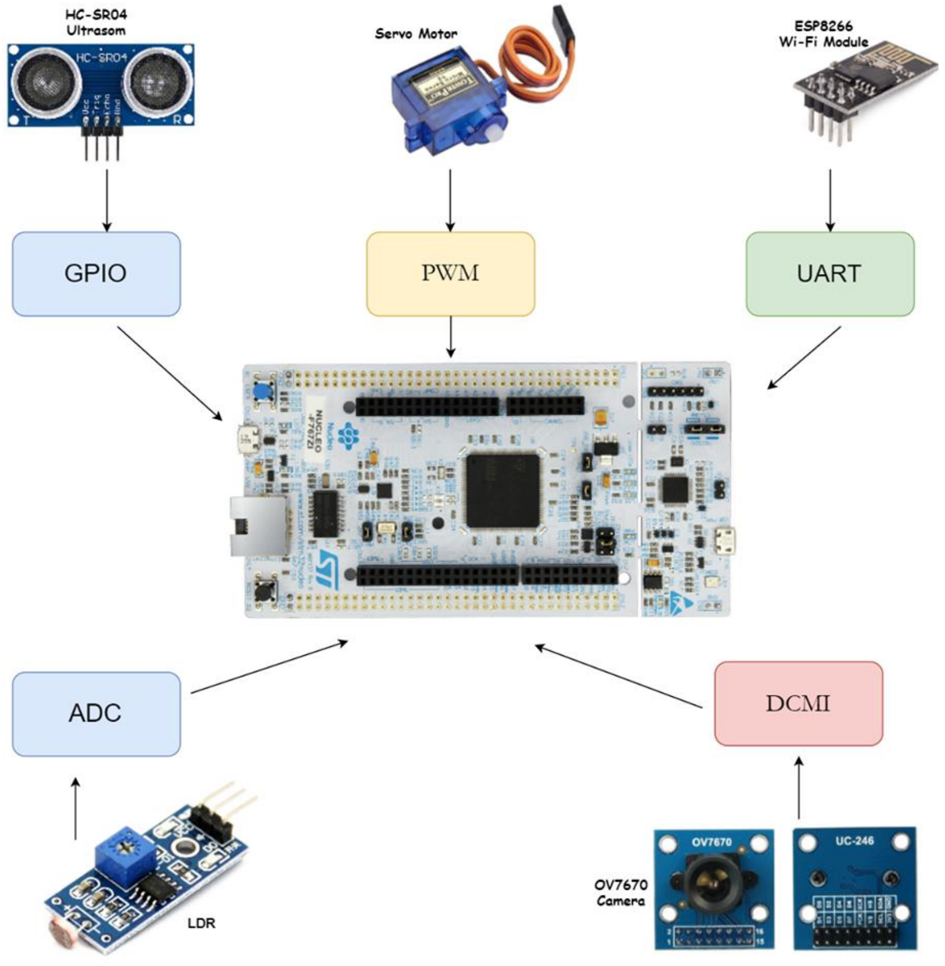


*Figure 27 - Hardware Peripherals and Communication*

## 2.1 PWM Protocol

Pulse Width Modulation is a method of reducing the average power delivered by an electrical signal, by effectively chopping it up into discrete parts. The average value of voltage fed to the load is controlled by turning the switch between supply and load on and off at a fast rate. PWM is be used to control the servo motors, due to the need of a duty cycle.
By controlling the time that signal spend in high state over a consistent interval, the percentage of time ON relative to the percentage of time to off, is called by duty-cycle.

The signal can only in the range of 5V(high) or ground(low). The PWM generated by the board is not directly induced in the motor, due to the lack of power/current, so the mosfet actuates in here to counter that limitation.

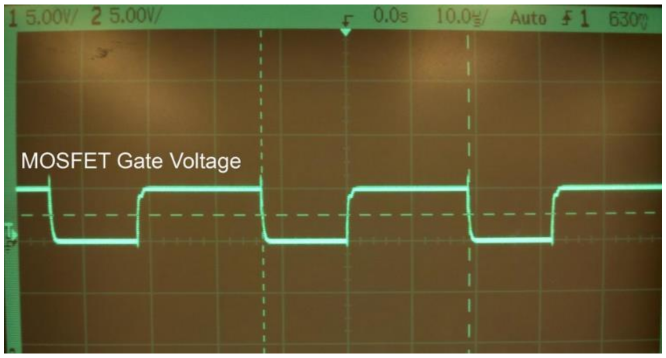


*Figure 28 - Mosfet PWM output wave*

## 2.2 UART protocol

The UART (Universal Asynchronous Receiver/Transmitter) is a peripheral used for full duplex serial communication that is presented in our board. It is a single LSI (large scale integration) chip designed to perform asynchronous communication. This device sends and receives data from one system to another system.

*www.codrey.com*

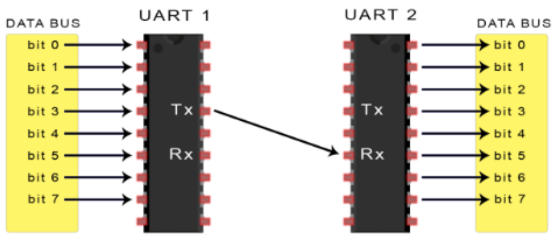


*Figure 29 - UART connectioin logic*

In UART communication, two UARTs communicate directly with each other. Basically, the transmitting UART converts parallel data into a serial form, and the destination receiving UART converts the serial form to a parallel data. The data flows from the Tx to the Rx pin of the receiving UART.

UARTs transmit data asynchronously, which means there is no clock signal to synchronize the output of bits from the transmitting UART to the sampling of bits by the receiving UART. Instead of a clock signal, the transmitting UART adds “start” and “stop” bits to the data packet being transferred. These bits define the beginning and end of the data package so the receiving UART knows when to start reading the bits.

In our system will be used to communication between the board and the wi-fi module.

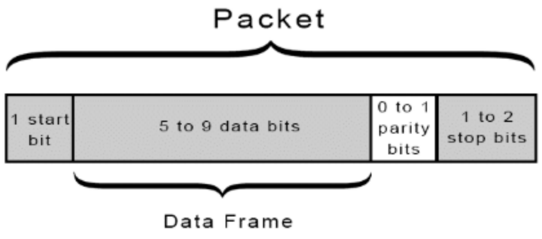


*Figure 30 - UART data packet*

## 2.3 DCMI and I2C protocol

The digital camera interface (DCMI) is a synchronous parallel data bus. It allows easy integration and easy adaptation to specific requirements of an application. The DCMI connects with 8-, 10-, 12- and 14-bit CMOS camera modules and supports a multitude of data formats.

*www.st.com*

**Camera module interconnect:**

To implement a camera on our board we need this protocol to achieve a communication between our board and camera. The camera interface works as bridge allowing the image sensor to connect to an embedded system and send or receive signals. The signals transferred between a camera and an embedded system are mainly:

- control signals
- image data signals
- power supply signals
- camera configuration signals.

Depending on the manner of transferring data signals, the camera interface is divided into two types: **parallel** and **serial interfaces**.

The camera module is connected to the DCMI through three types of signals:

- DCMI -> clock and data signals
- I2C -> configuration signals

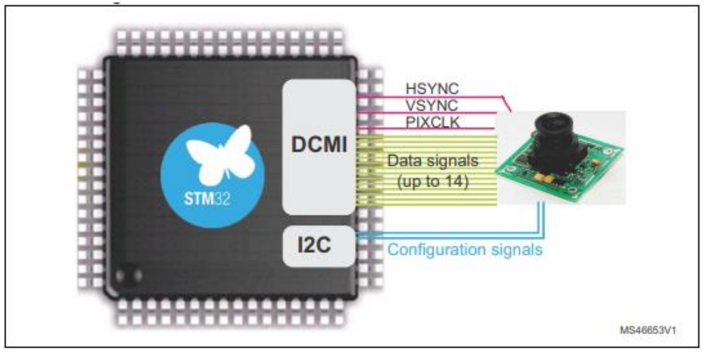


*Figure 31 - DCMI - Camera connection*

**Capture modes:**

The DCMI supports two types of capture: snapshot (a single frame) and continuous grab (a sequence of frames). The continuous grab is the one that we will use.

**DCMI interrupts:**

Five interrupts can be generated:

- **IT_LINE** indicates the end of line;
- **IT_FRAME** indicates the end of frame capture;
- **IT_OVR** indicates the overrun of data reception;
- **IT_VSYNC** indicates the synchronization frame;
- **IT_ERR** indicates the detection of an error in the embedded synchronization codes order (only in embedded synchronization mode).

All interrupts can be masked by software. The global interrupt DCMI it's the "**OR"** logic function of all the individual interrupts. The DCMI interrupts are handled through three registers:

- **DCMI_IER**: read/write register allowing the interrupts to be generated when the corresponding event occurs;
- **DCMI_RIS**: read-only register giving the current status of the corresponding interrupt, before masking this interrupt with the DCMI_IER register (each bit gives the status of the interrupt that can be enabled or disabled in the DCMI_IER register);
- **DCMI_MIS:** read-only register providing the current masked status of the corresponding interrupt, depending on the DCMI_IER and the DCMI_RIS registers.

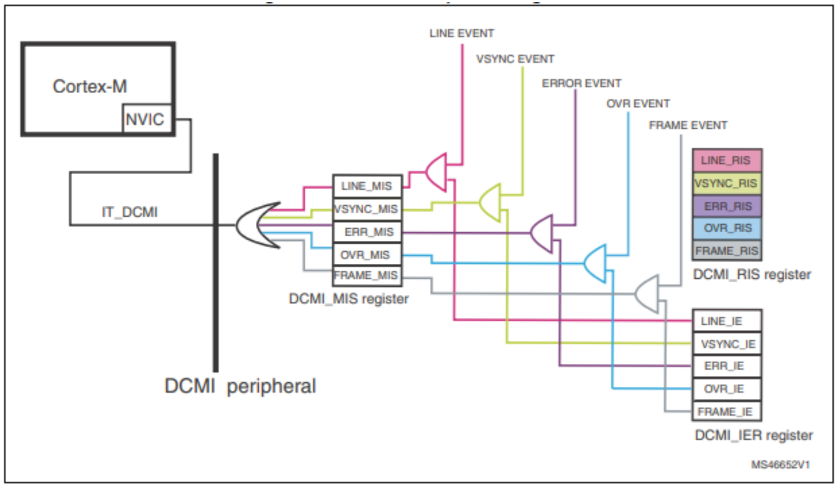


*Figure 32- DCMI registers overview*

**DCMI configuration**:

When selecting a camera module to interface with STM32 MCUs, the user should consider some parameters like: the pixel clock, the supported data format and the resolutions. To correctly implement his application, the user needs to perform the following configurations:

- Configure the GPIOs.
- Configure the timings and the clocks.
- Configure the DCMI peripheral.
- Configure the DMA.
- Configure the camera module:
    – configure the I2C to allow the camera module configuration and control
    – set parameters such as contrast, brightness, color effect and data format.

# 3. Software Specification

In this chapter, be define and mention specific software modules, tools and applications, as it is necessary to have a description of the software that's going to be developed. As so, this chapter should provide an inside out of the work and structure that needs to be followed to have a good implementation of the project.

**Tools**
In order to develop the project, some tools are needed for training the neural network, an IDE to
code our program (and its language and libraries), and other applications. For this project it will be used the following ones:

- Keil _Vision5 – IDE (Interface Development Environment) for programming for the STM board with capability for debug.
- ST-LINK - Debugger on chip tool, that allows to follow the state of the board while debugging
- the code.
- STM32CubeMX - Configuring tool for STM board. Generates code for the initialization and
- some configurations of the board's peripherals.
- MATLAB - Tool used to train the neural network

**COTS**
Commercial-Off-The-Shelf (COTS) are packaged solutions which are then adapted to satisfy the needs of the project. Since there are some APIs needed to develop the project, some COTS are essential. The ones used on this project are:

- FreeRTOS - real time operation system;
- CMSIS-NN - for AI libraries;
- HAL - abstraction layer for peripheral drivers;
- CMSIS-DSP - for signal processing libraries.

## 3.1 FreeRTOS

A Real Time Operating System is an operating system that is optimized to be used in embedded/real time applications. Their primary objective is to ensure a timely and deterministic response to events.

Using a real time operating system allows a software application to be written as a set of independent tasks. Each task is assigned a priority and it is the responsibility of the Real Time Operating System to ensure that the task with the highest priority is the one that is running.

To avoid race conditions, the OS also provides methods for thread synchronization like mutexes, semaphores and software timers. The functions related to this topic are:

| Functions | Purpose |
|---|---|
| **xTaskCreate** | Creates a new task and add it to the list of tasks that are ready to run. |
| **xQueueCreate** | Creates a new queue. |
| **xQueueSendToBack** | Is used to send data to the back (tail) of a queue |
| **xQueueRecieve** | Is used to receive (read) an item from a queue |
| **xQueueSendFromISR** | Has the same functionality as the xQueueSendToBack but is used to a interrupt |
| **xQueueRecieveFromISR** | Has the same functionality as the xQueueReceive but is used to a interrupt |
| **xSemaphoreCreateBinary** | Creates a binary semaphore. |
| **xSemaphoreGive** | Creates a specific type of semaphore, the mutex. |
| **xSemaphoreCreateMutex** | Takes a semaphore or mutex |
| **xSemaphoreTake** | Gives the mutex or semaphore. |

| | |
|---|---|
| **vTaskNotifyGiveFromIsr** | Is a function intended for use when an RTOS task notification value is being used as a light weight and faster binary or counting semaphore alternative. FreeRTOS semaphores are given from an interrupt using the xSemaphoreGiveFromISR() API function, vTaskNotifyGiveFromISR() is the equivalent that instead uses the receiving RTOS task's notification value. |
| **ulTaskNotifyTake** | Allows a task to wait in the Blocked state for its notification value to be greater than zero, and either decrements or clears the task's notification value before it returns. |
| **vTaskStartScheduler** | Starts the RTOS scheduler |

*www.freertos.org*

*Table 16 - FreeRTOS functions*

## 3.2 CMSIS-DSP

The idea behind CMSIS is to provide a consistent and simple software interface to the processor for interface peripherals, real-time operating systems, and middleware, simplifying software re -use, reducing the learning curve for new microcontroller developments and reducing the time to market for new devices (CMSIS library comes with ST firmware).

The CMSIS-DSP library is designed for Cortex-M processors and it provides optimized functions for digital signal processing such as:

- Basic mathematical functions with vector operations;
- Fast mathematical functions, like sine and cosine;
- Complex mathematical functions like calculating magnitude;
- Filtering functions like FIR or IIR;
- Matrix computing functions;
- Transform functions like FFT;
- Controller functions like PID controller;
- Statistical functions like calculating minimum or maximum;
- Support functions like converting from one format to another;
- Interpolation function.

## 3.3 CMSIS-NN

CMSIS-NN is a collection of optimized neural network functions for ARM Cortex-M core microcontrollers enabling neural networks and machine learning being pushed into the end node of IoT applications.

It has implemented popular neural network layer types, such as convolution, depth separable convolution, fully-connected, polling, and activation. With its utility functions, it is also possible to construct more complex NN modules, such as LSTM and GRU.
The weights and biases will first be quantized to 8 bit or 16-bit integers then deployed to the microcontroller for inferencing.

Neural network inference based on CMSIS-NN kernels claimes to achieve 4.6X improvement in runtime/throughput and 4.9X improvement compared to baseline implementation. The best performance was achieved by leveraging SIMD instructions features of the CPU to improve parallelism available for Cortex-M4 and Cortex-M7 core microcontrollers although reference implementation for Cortex-M0 and Cortex-M3 is also available without DSP instructions.

*www.dlology.com*

The library is divided into a number of functions each covering a specific category:

- Neural Network Convolution Functions
- Neural Network Activation Functions
- Fully-connected Layer Functions
- Neural Network Pooling Functions
- Softmax Functions
- Neural Network Support Functions

The library has separate functions for operating on different weight and activation data types including 8-bit integers and 16-bit integers.

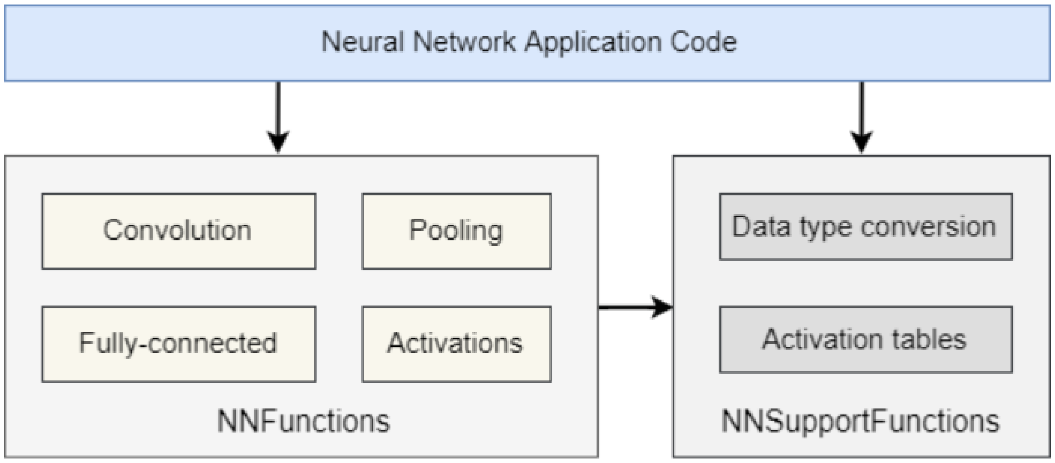


*Figure 33 - Neural Network Block Diagram*

## 3.4 HAL

The HAL driver layer provides a generic multi instance simple set of APIs (application programming interfaces) to interact with the upper layer (application, libraries and stacks). It is composed of generic and extension APIs. It is directly built around a generic architecture and allows the built-upon layers, such as the middleware layer, to implement their functions without knowing in-depth how to use the MCU. This structure improves the library code reusability and guarantees easy portability to other devices.

The HAL drivers include a complete set of ready-to-use APIs which simplify the user application implementation. As an example, the communication peripherals contain APIs to initialize and configure the peripheral, to manage data transfers based on polling, to handle interrupts or DMA, and to manage communication errors.

Cube uses the STM32 hardware abstraction layer (HAL) library to create the initialization code, which makes it a lot easier to migrate between STM32 microcontrollers if needed.

# 4.Task Overview

To achieve a proper functioning of our system we need to establish the threads communication and how they reach synchronization. Therefore, this diagram includes the types of synchronization between the tasks, which are the Mutex, the Semaphore and the crossed information through the different tasks (Queue).

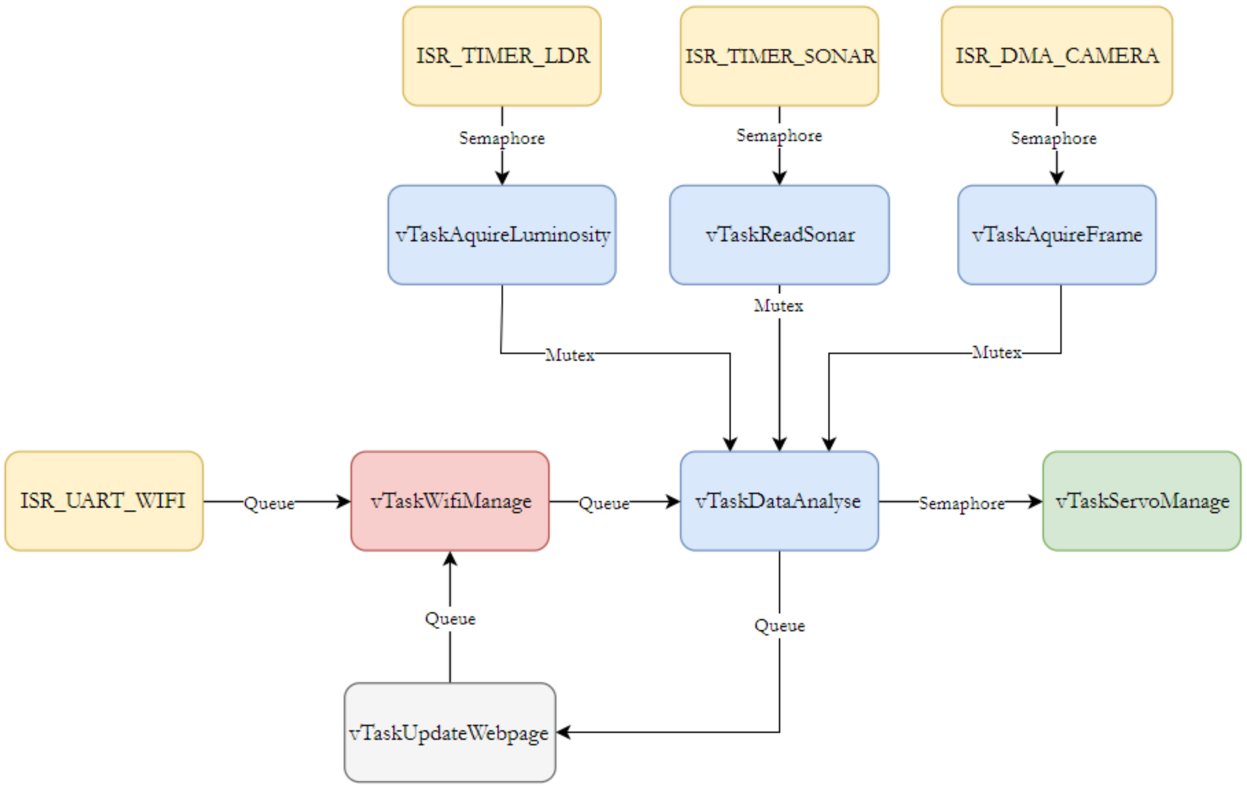


*Figure 34 - Task Overview Diagram*

## 4.1. Tasks

**vTaskAquireLuminusity**: This task is responsible to acquire the value of luminosity through ADC and saves it to a structure for later analysis.

**vTaskReadSonar**: This task is responsible to read sonar value and indicate the system if a object was detected. Whether an obstacle is detected the distance can be measurement and saves the measurement into a structure for later analysis.

**vTaskAquireFrame**: This task is responsible to acquire frames of camera, endode the pixel image and save the frame on the memory.

**vTaskDataAnalyse:** After all of data is acquired, the system gathers and analyze all the data, signals to update the Webpage, and actuate on the servos motors based on the values read/choosen.

**vTaskUpdateWebpage:** This task will receive the analyzed information and update the file that contain the sensor's information on the webpage.

**vTaskServoManage:** This task will be responsible for actuate and set the angle of the Servo Motors.

**vTaskWifiManage:** this task will manage all the communications via WiFi, by sending requests and information to receive and analyze the incoming data.

## 4.2 ISR

**ISR_TIMERLDR:** This interrupt is responsible to alert TaskAquireLuminusity that can make the reading of the adc and acquire the luminosity values.

**ISR_DMACamera:** This ISR is responsible for alerting the vTaskAcquireFrame that can acquire a frame.

**ISR_TIMERSonar:** This ISR is responsible for to signal the gathering of samples of the sonar.

**ISR_UART_WiFi:** This ISR is responsible for alerting the system that a message was received from the webpage.

## 4.3 Task Synchronization

**Sem_ActivateServo:** This semaphore unlocks the *vTaskServoManage,* allowing the actuation of servo-motors.

**Sem_ReadSonar:** This semaphore unlocks the *vTaskAquireLuminusity* when Sonar timer trigger.

**Sem_ReadLdr:** This semaphore unlocks the *vTaskReadSonar* when Ldr timer trigger.

**Sem_ReadCamera:** When the DMA finish reading from the camera it allows the TaskAquireFrame to read from the buffer.

**Mutex_Luminosity:** This mutex is responsible for the luminosity data protection, that is a shared resource between the *vTaskAquireLuminusity* and *vTaskDataAnalyse*.

**Mutex_Sonar:** This mutex is responsible for the sonar variables protection, that is a shared resource between the *vTaskReadSonar* and *vTaskDataAnalyse*.

**Mutex_Frame:** This mutex is responsible for frame protection, that is a shared resource between the *vTaskAquireFrame* and *vTaskDataAnalyse*.

## 4.4 Queue

**Queue_DataToAnalyze:** Send the inputs of user from *vTaskWifiManage* task to *vTaskDataAnalyse* task via queue.

**Queue_UpdateWebpage:** Send the analyzed data from *vTaskDataAnalyse* task to *vTaskUpdateWebpage* task via queue.

**Queue_WifiManage:** Send the file that build webpage from *vTaskUpdateWebpage* task to *vTaskWifiManage* task via queue.

**Queue_RxWIFI:** Send data received by the ISR_UART_WiFi interrupt to the *vTaskWifiManage* task via queue.

## 4.5 Task priority:

In a system with a great number of tasks, each with its purpose, it's necessary to prioritize them in order to achieve our goal.

Not every task is equally important, some more than others. It is necessary to identify what is the most important task at any moment and give those tasks more attention, energy, and time. For our system we design these priority task pyramid diagram below. The reading sonar task and acquire frames task, were chosen as the most important tasks because in case a object appears instantly, or get closer more and more, the system must have to know that, and the camera must capture this movement to alert the User. The lower layer is presented by acquisition of environment luminosity just because is a task that don´t have most importance relatively to the purpose of our system.

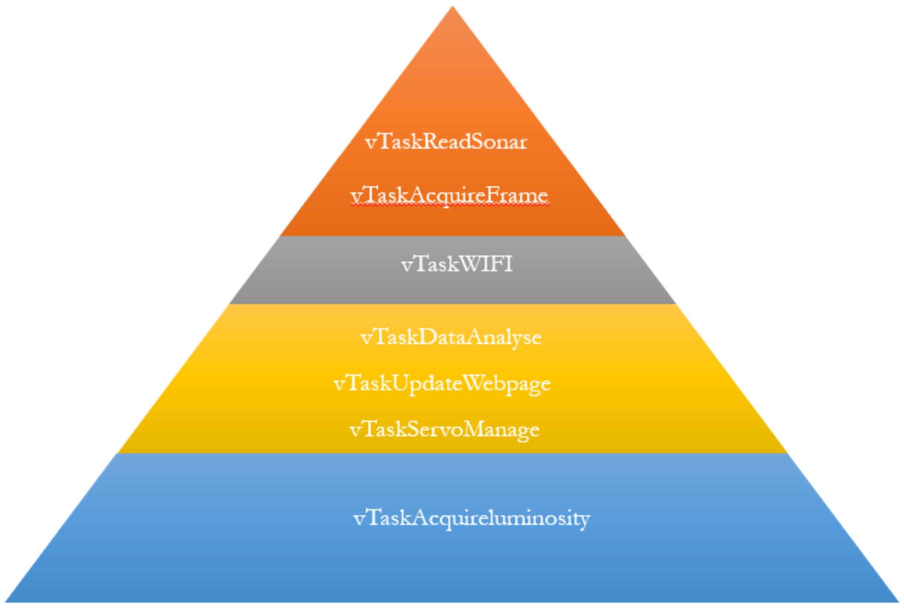


*Figure 35 -Task priority*

## 4.6 Flowcharts

### System Initialization

To start the project, the system first initializes all the process. In the diagram below is represented the task initialization process, where it initializes all the modules in a specific designed order. This is the first instruction accomplish in our system.

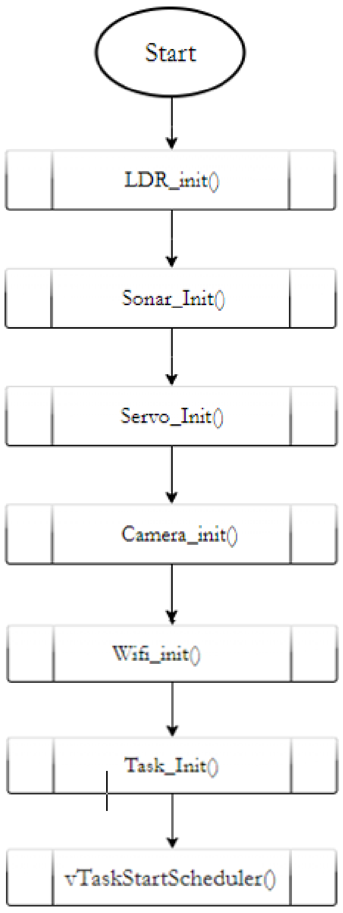


*Figure 37 - Initialization Flowchart*

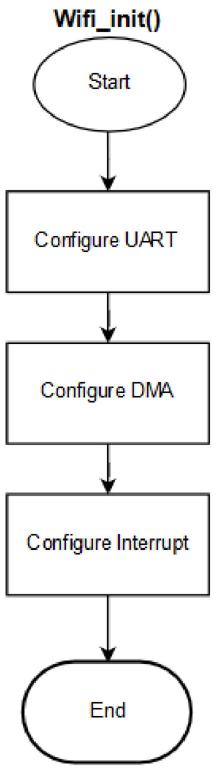


*Figure 38 - WiFi_init Flowchart*

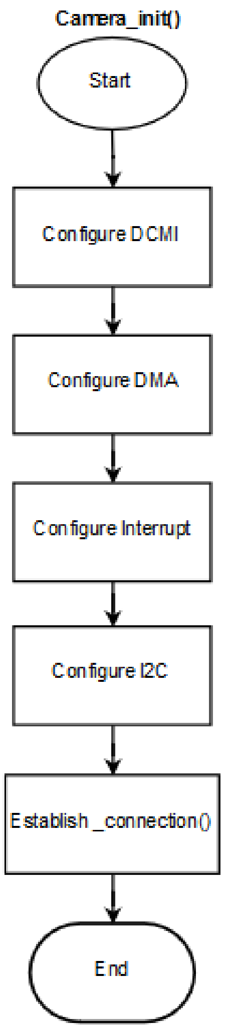


*Figure 36 - Camera_init() Flowchart*

**LDR_ISR and vTaskAquireLuminosity**

To gather all the parameters related to the environment the system must measure the luminosity. For that, there is this task, to acquire luminosity value. The interrupt reads the ADC value and gives the semaphore signal for the vTaskAquireLuminosity.
The luminosity will be saved in a structure that contains all the values of environment characteristics.

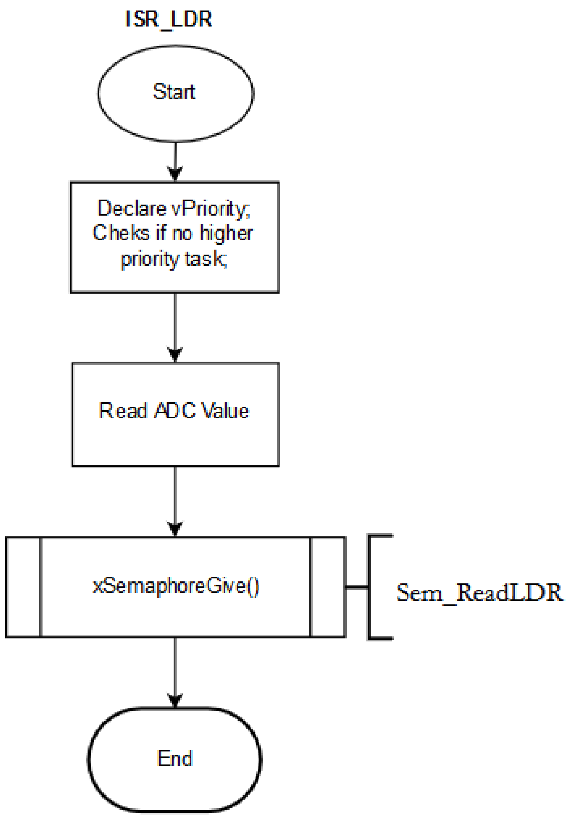


*Figure 39 - ISR_LDR Flowchart*

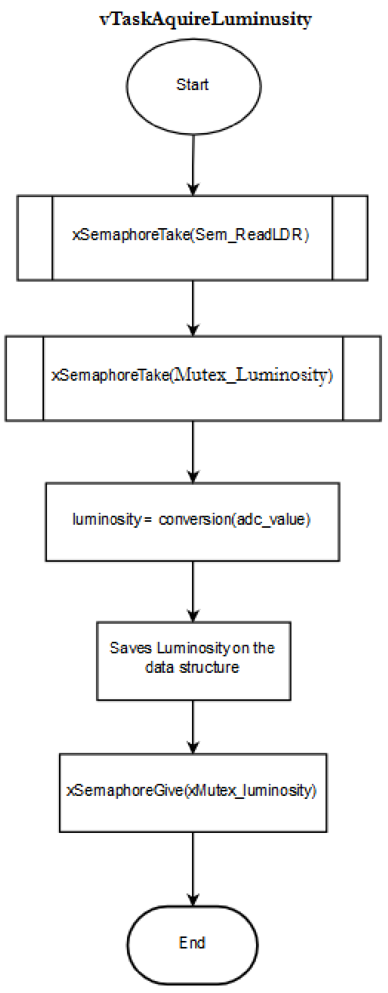

vTaskAquireLuminusity
Start
xSemaphoreTake(Sem_ReadLDR)
xSemaphoreTake(Mutex_Luminosity)
luminosity = conversion(adc_value)
Saves Luminosity on the data structure
xSemaphoreGive(xMutex_luminosity)
End

*Figure 40 - vTaskAquireLuminosity Flowchart*

**Sonar_ISR and vTaskReadSonar**

The most important action in our system is the detection of objects and study how much distant are from our device. This task will be responsible for that and is described more detailed in this flowchart bellow.
The interrupt reads the output of the sonar sensor and gives the semaphore signal of the vTaskReadSonar. In the vTaskReadSonar, it calculates the distance (variable protected by a mutex) and saves its value. If the distance exceeds 4 meters it defines it as no object detected.

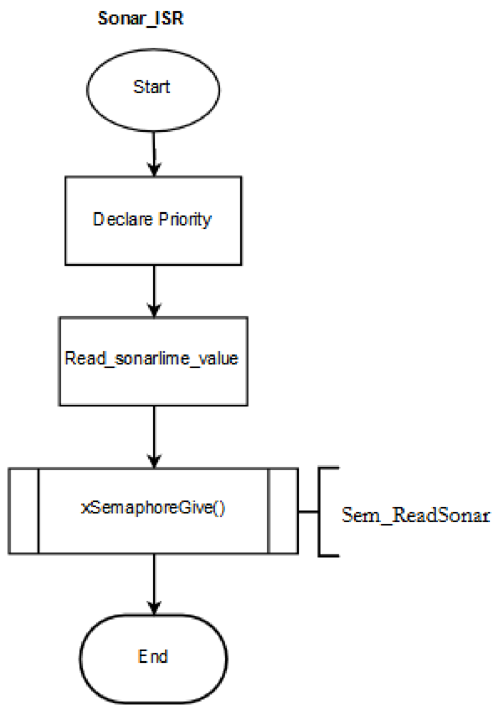


*Figure 42 - Sonar_ISR Flowchart*

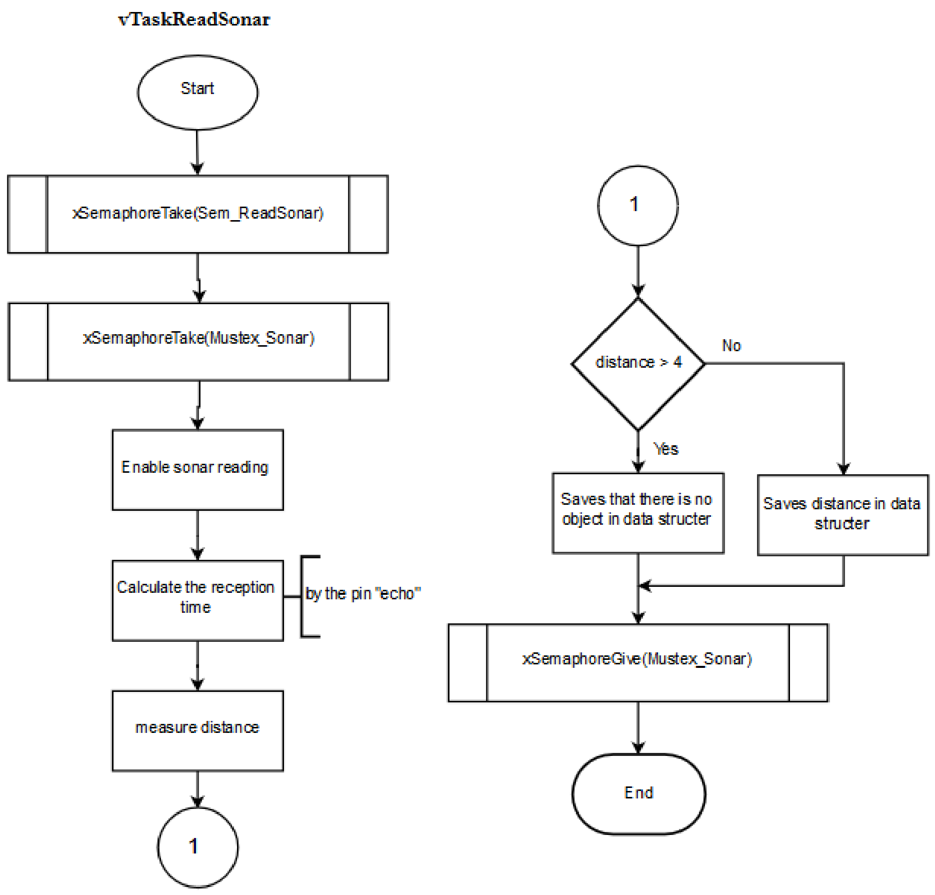

vTaskReadSonar
Start
xSemaphoreTake(Sem_ReadSonar)
xSemaphoreTake(Mustex_Sonar)
Enable sonar reading
Calculate the reception time
by the pin "echo"
measure distance
1
1
distance > 4
No
Yes
Saves that there is no object in data structer
Saves distance in data structer
xSemaphoreGive(Mustex_Sonar)
End

*Figure 41 - vTaskReadSonar Flowchart*

**ISR_DMA_CAMERA and vTaskAcquireFrame**

For a better environment perception, we use a camera to visualize in real time the environment. In the flowchart bellow shows the interrupt enabling the semaphore signal for the vTasjaquireFrame, and in the task, it acquires a capture and saves it on the structure (instructions protected by the mutex):

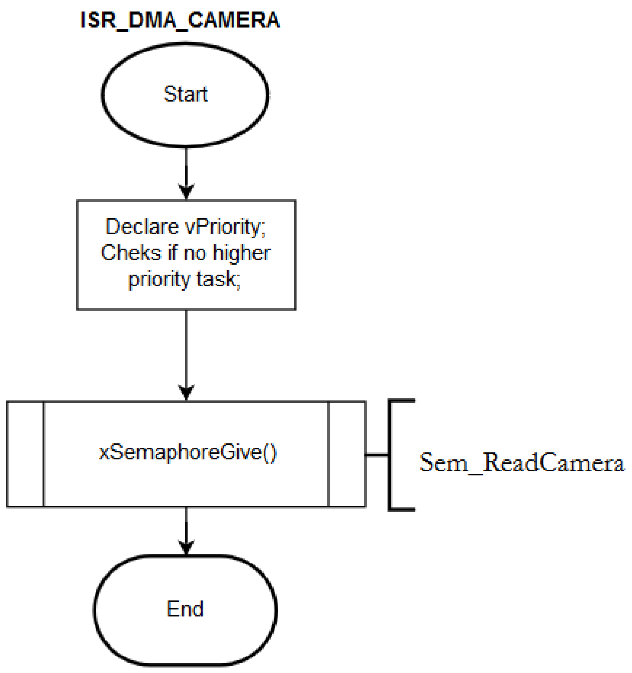


*Figure 44- ISR_DMA_CAMERA Flowchart*

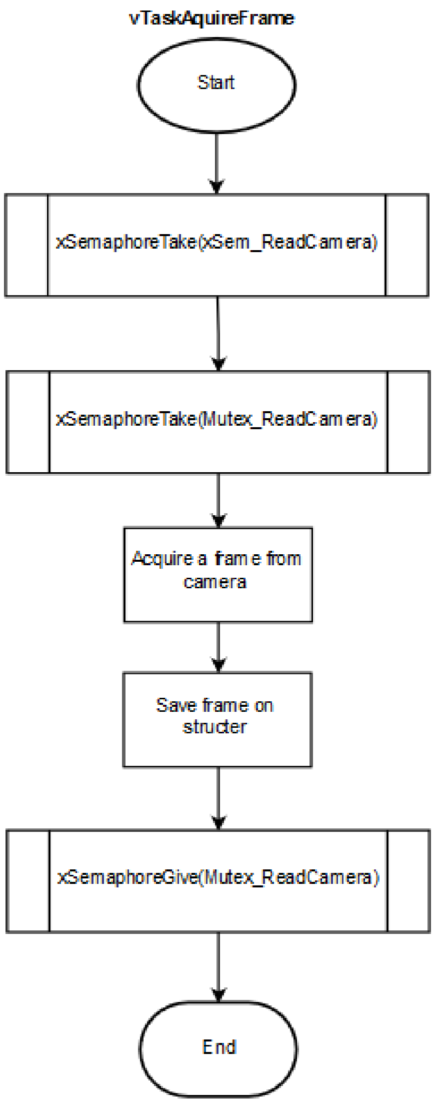

vTaskAquireFrame
Start
xSemaphoreTake(xSem_ReadCamera)
xSemaphoreTake(Mutex_ReadCamera)
Acquire a frame from camera
Save frame on structer
xSemaphoreGive(Mutex_ReadCamera)
End

*Figure 43- vTaskAquireFrame Flowchart*

**ISR_UART_WiFi, vTaskAquireFrame and VTask WiFiManage**

Here are presented all the work flow of the WiFi communication (transmitting and receiving data) between the STM and the Webpage interface. Starting on the ISR, the interrupt activates when there's data to the wifi module and it gathers the UART characters and sends a queue message to the vTaskWiFiManage.
The vTaskUpdateWebpage, when received the queue message to update, it gathers the analyzed data and compact it and signals the vTaskWiFiManage to transmit the data to the Webpage via the module.
The vTaskWifiManage is responsible to effectively perform the communication, based on the queue message received.

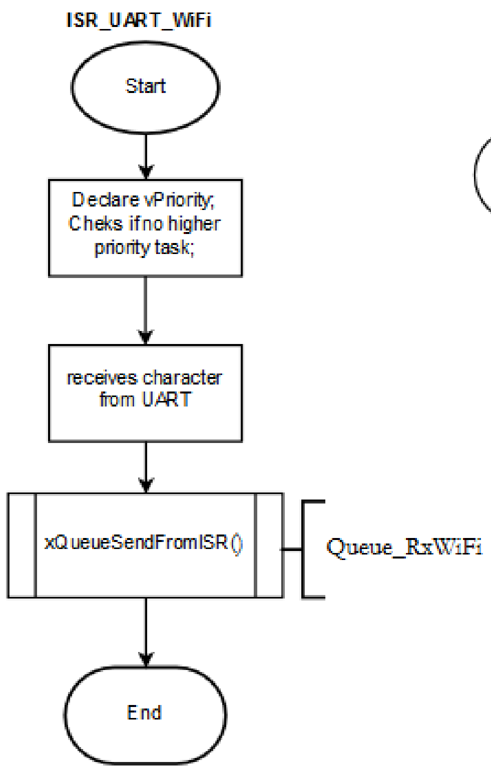

ISR_UART_WiFi
Start
Declare vPriority; Cheks if no higher priority task;
receives character from UART
xQueueSendFromISR()
Queue_RxWiFi
End

*Figure 46- ISR_UART_WiFi Flowchart*

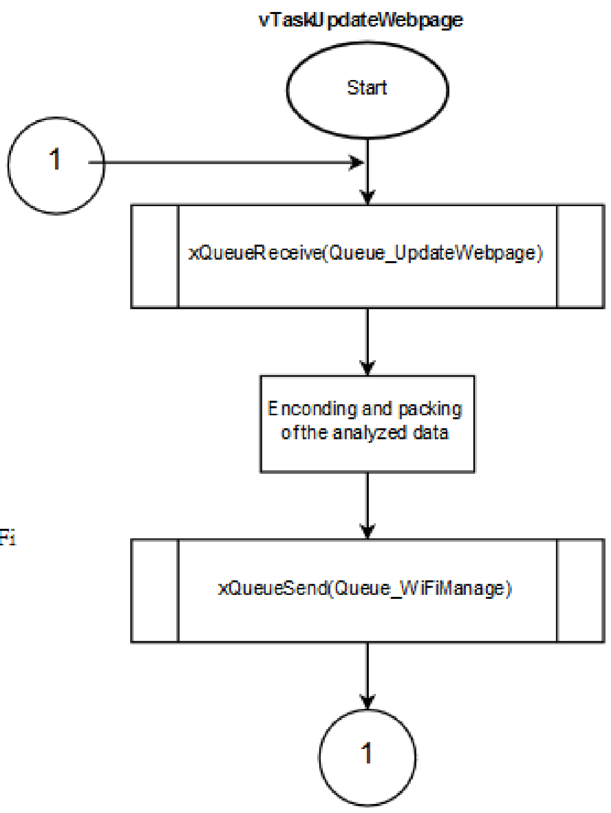

vTaskUpdateWebpage
Start
1
xQueueReceive(Queue_UpdateWebpage)
Enconding and packing of the analyzed data
xQueueSend(Queue_WiFiManage)
1

*Figure 45 - v TaskUpdateWebpage Flowchart*

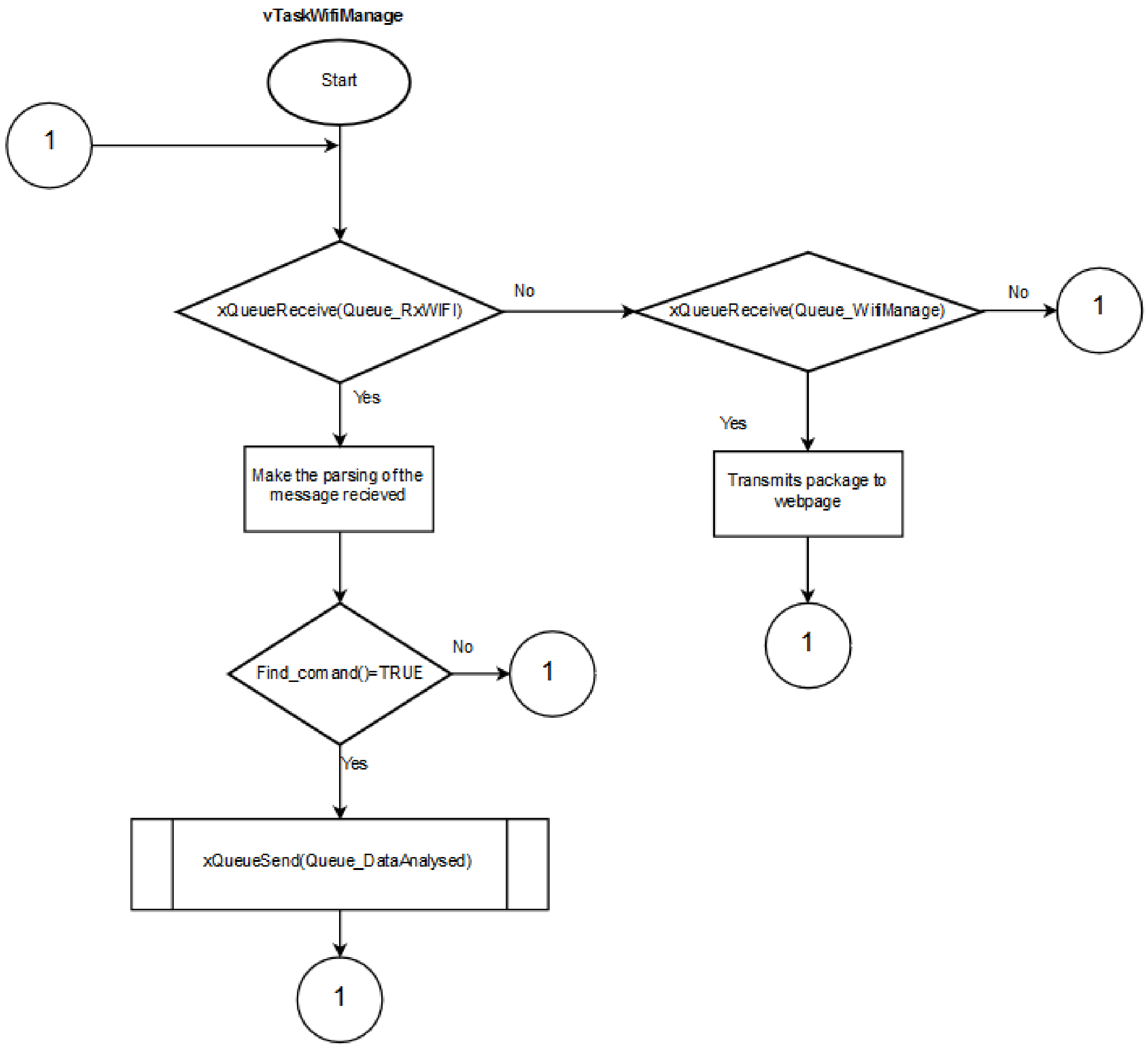
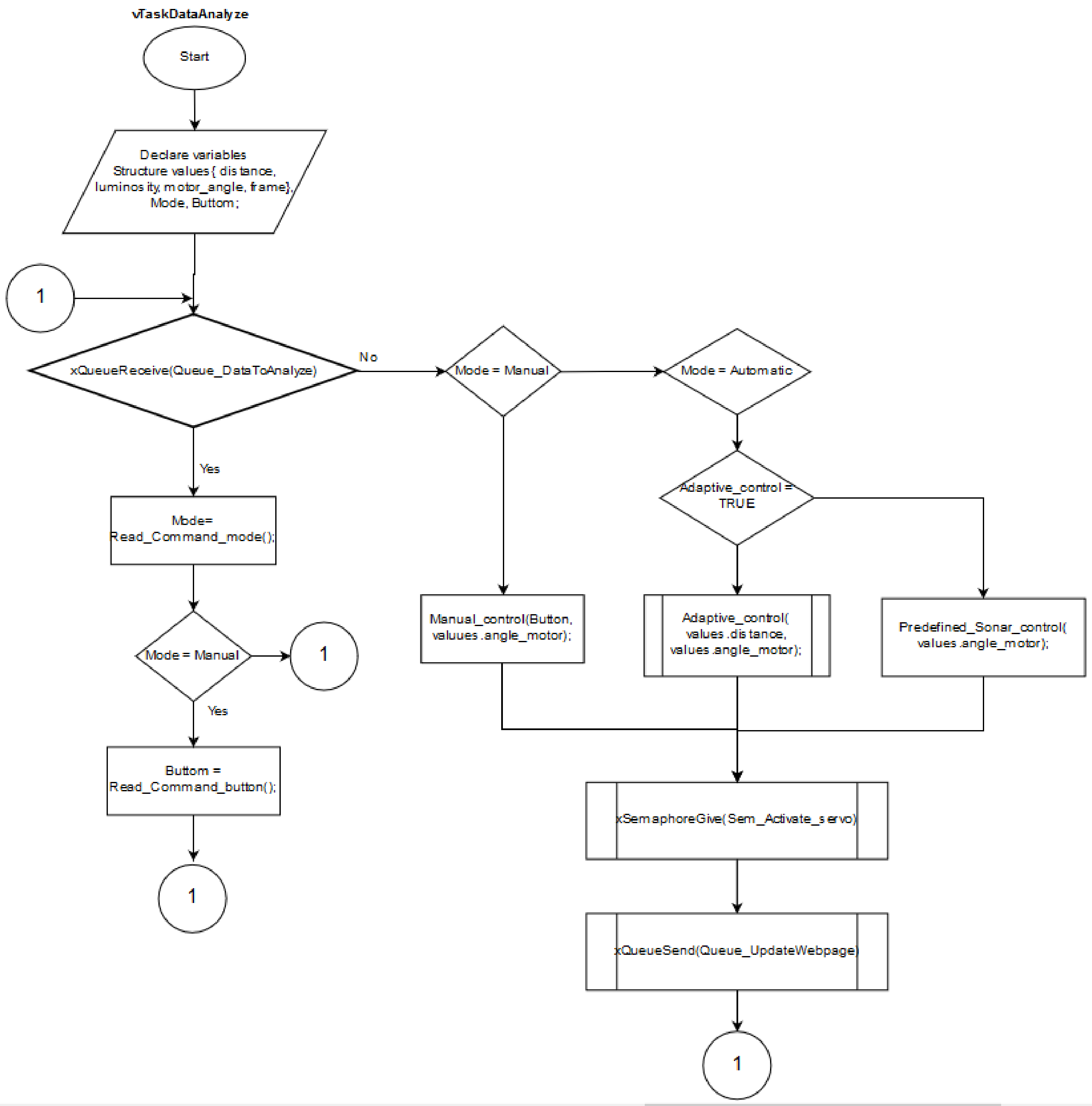

vTaskWifiManage
Start
1
xQueueReceive(Queue_RxWIFI)
No
xQueueReceive(Queue_WifiManage)
No
1
Yes
Yes
Make the parsing of the message recieved
Transmits package to webpage
1
Find_comand()=TRUE
No
1
Yes
xQueueSend(Queue_DataAnalysed)
1

*Figure 47 - vTaskWiFiManage Flowchart*

**vTaskDataAnalyze**

This is the main task of the system, focused on analyzing the data and deducting the control to be used in practice. First, it's declared the variables (such as the global data values structure). Then, if it receives a queue message signaling, it reads the control mode or the button, send by the Webpage interface. Else, if there is no queue message, it checks the mode in to operate: if manual, it sends the direction (based on the buttons) to actuate to the servos; if automatic and if to apply adaptive control it will apply the advanced algorithm to determine the presence/location of a object and calculate the direction to apply on the motors to follow the camera. If not adaptive control, it applies a pre-defined a control, to just get a 180º sonar view of the environment.



vTaskDataAnalyze
Start
Declare variables
Structure values{ distance, luminosity, motor_angle, frame},
Mode, Buttom;
1
xQueueReceive(Queue_DataToAnalyze)
No
Yes
Mode = Manual
Mode = Automatic
Adaptive_control = TRUE
Mode= Read_Command_mode();
Mode = Manual
1
Yes
Buttom = Read_Command_button();
1
Manual_control(Button, valuues.angle_motor);
Adaptive_control( values.distance, values.angle_motor);
Predefined_Sonar_control( values.angle_motor);
xSemaphoreGive(Sem_Activate_servo)
xQueueSend(Queue_UpdateWebpage)
1

*Figure 48- vTaskDataAnalyze Flowchart*

**tActuate_Servo**

The tActuate_Servo is the actuation of the servo-motor with the specific PWM value, based on the calculated control. In the first part, this thread will initiate when it breaks from the semaphore waiting state. When a signal for the semaphore is taken/received, it receives the calculated control from the task vTaskDataAnalyze and applies the specific PWM value that defines a new angle of the camera.

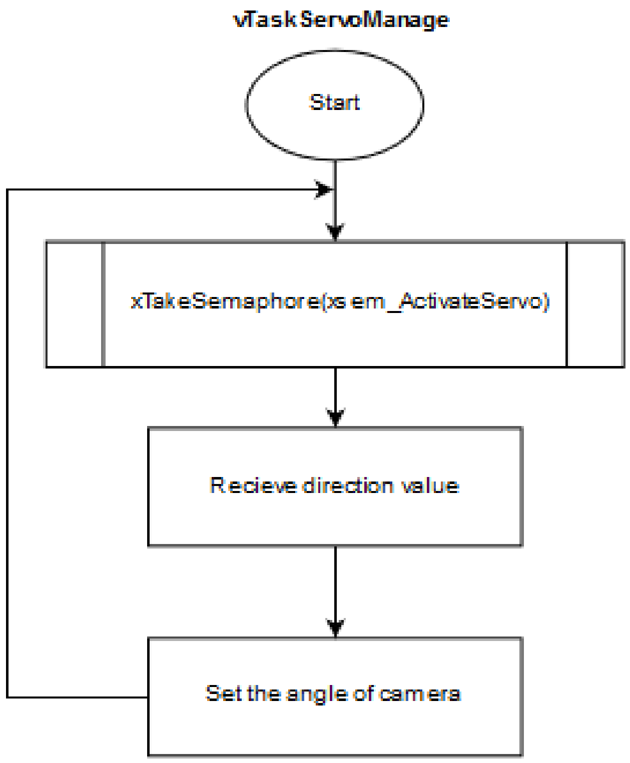


*Figure 49 - vTaskServoManage Flowchart*

## IV. Gantt Diagram

Based on the Waterfall method we can schedule the project in five major entities: Documentation, Analysis, Design, Implementation and Testing. Estimating each task, test, and proceeding, it concludes on a 4 months long project.

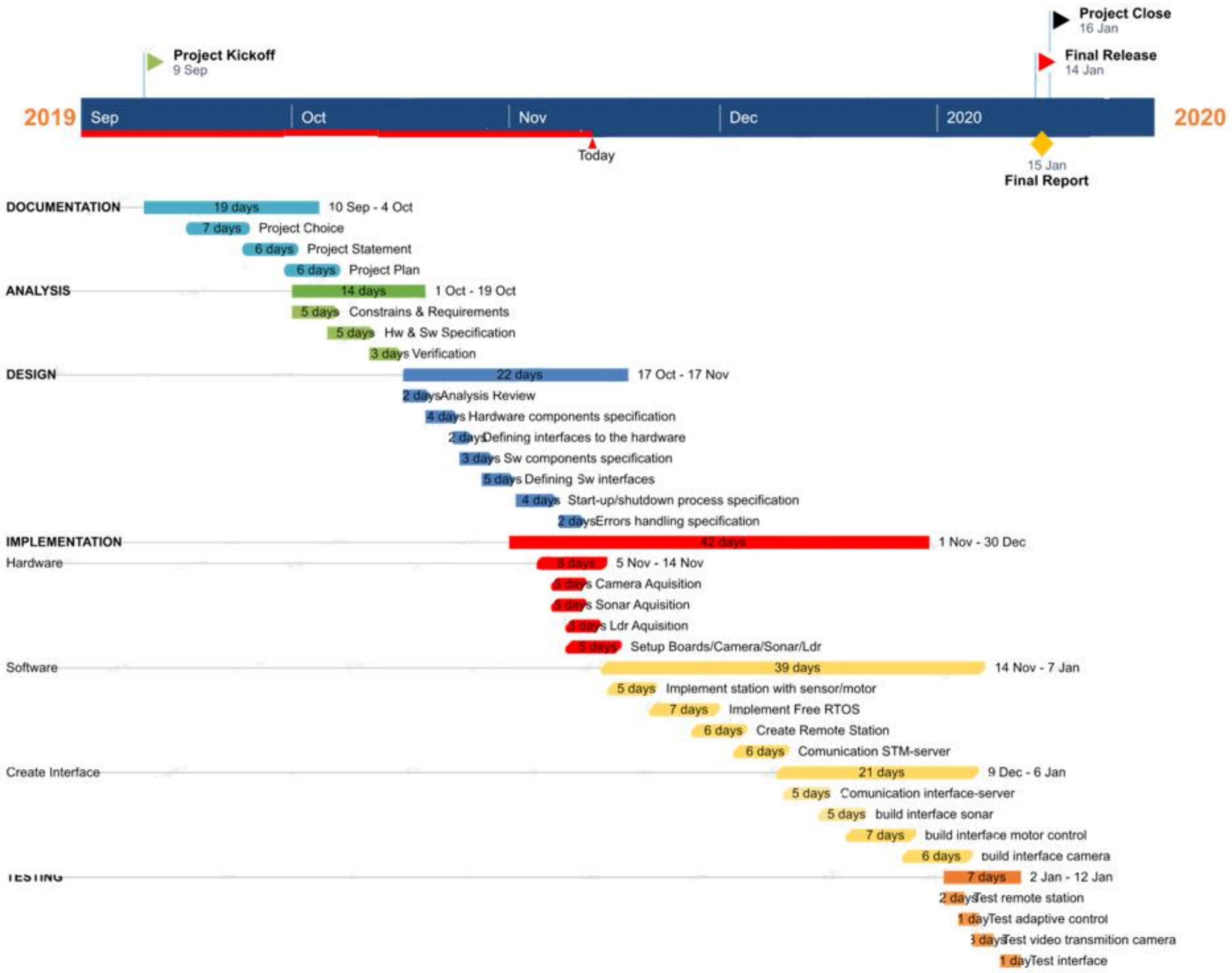


Figure 50 - Gantt Diagram